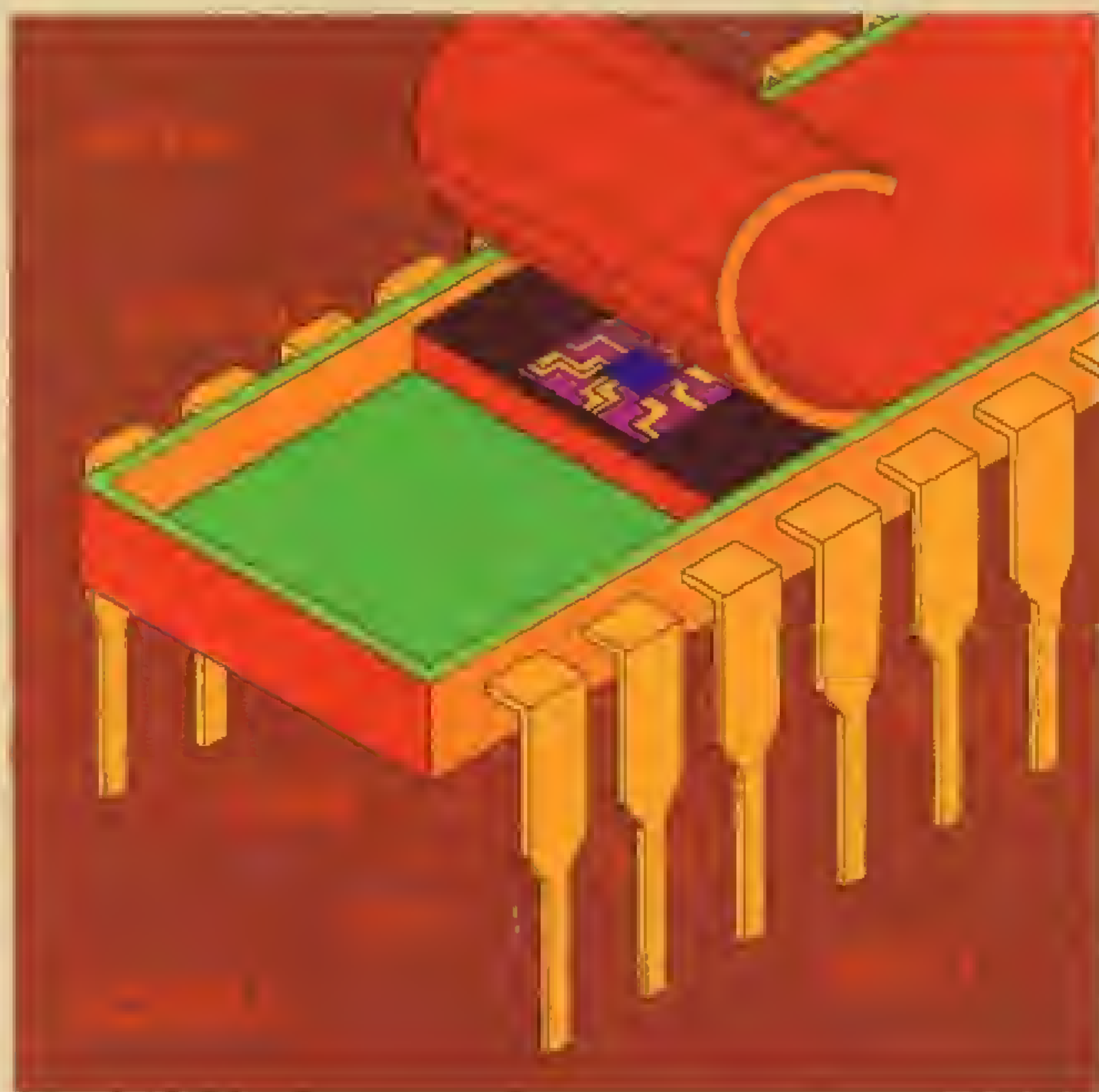
Apple II

Apple II Monitors Peeled



10260 Bandley Drive
Cupertino, California 95014
(408) 996-1010

APPLE Product #D2L0013
950-0018

Printed in USA

Apple II

Apple II Monitors Peeled

TABLE OF CONTENTS

PREFACE vii

INTRODUCTION vii

OVERVIEW viii

CHAPTER 1

MEMORY ALLOCATION 1

1 Monitor Usage Memory Map
2 RAM Memory Allocation by Address
3 Page Zero
3 Page Zero Fields
13 Pages One through Three
13 Page One ($Ø1ØØ - Ø1FF)
13 Page Two ($Ø2ØØ - Ø2FF)
13 Page Three ($Ø3ØØ - Ø3FF)
14 Page Three Address Table
14 Pages Four through Seven & Eleven
15 Screen Memory Address Table
16 Peripheral Controller Work Areas

CHAPTER 2

INPUT AND OUTPUT 17

17 Keyboard Input Division of Labor
18 Table of Routines
21 Calls to Keyboard Input Routines
21 Table of Keyboard Input Calls
24 KEYIN Routine Replacement
25 Keyboard Input Monitor Routine
26 Address Table 1 - Character Input
27 Address Table 2 - Line Input
28 Overview - Text Output to the Screen
29 Output within the Scroll Window
31 Page Zero Fields
33 Scroll Window Output Routines
34 Screen Format Control by Routine
35 Screen Format Control by Poke/Store
36 Scroll Window Data Manipulations
36 Address Table
38 Cursor Position Control
39 Address Table
41 General Text to the Screen
41 Address Table
43 Control Characters
43 Output without the Scroll Window
45 Address Table
46 Applesoft Sample Program
47 Secondary Display Areas
47 Copy Primary to Secondary
48 Set BASL,H for Secondary Display Page
48 Address Table
49 Direct Control Addresses
5Ø Integer BASIC Sample Program
5Ø Applesoft Sample Program

CHAPTER 3

INTERRUPT PROCESSING

53

53 Interrupt Processing
54 NMI Interrupt
54 RESET Interrupt Support
55 IRQ/BRK Interrupt Handling
55 RESET Interrupt - Old Monitor
56 Address Table
56 RESET Interrupt - Autostart Monitor
57 Initialize System Configuration
57 Cold/Warm Determination
58 Power-On Initialization
58 System Restart
58 RESET Vector Modification by User
59 Address Table
61 IRQ/BRK Interrupts
61 IRQ/BRK Interrupt Recognition
61 IRQ Interrupt Handling
61 BRK Instruction Interrupt
62 BRK Instruction - Saving of Status
62 BRK Instruction - Old Monitor
62 BRK Instruction - Autostart Monitor
63 Address Table

CHAPTER 4

MISCELLANY

65

65 Machine Language Development Aids
65 Address Table
67 LORES Plotting
68 Page Zero Fields
68 Address Table
69 Data Manipulation Functions
69 Routines
69 Memory to Memory Move
70 Jump to Address with Registers Loaded
70 Increment Address Fields
70 Save 6502 Registers
70 Restore 6502 Registers
70 Multiply Two Byte Fields
71 Multiply Routine
72 Divide Four Byte Dividend by Two Byte Divisor
73 Establish a RESET Vector
73 Convert Hex Characters to Value for Use
73 Disassemble an Instruction
74 Address Table
75 Applesoft Sample Data Manipulation Program
76 Monitor Command Processor
76 Entering the Monitor Command Processor
77 Calling the Monitor Command Processor
77 Address Table
80 Applesoft Sample Program
80 Speaker use through the Monitor
81 Address Table
81 Cassette Tape Input and Output
82 WRITE
82 READ
82 Cassette Input/Output Internal Routines
83 HEADR
83 RD2BIT
83 RDBIT
83 RDBYTE
84 WRBIT
84 WRBYTE
84 Paddles, Buttons and Annunciator I/O
85 Game I/O Hardware Address Table
85 WAIT Routine
86 WAIT Routine Delay Times
86 Use of Control-Y with Parameters
87 Paddle Interference - Sample Program
88 Registers for BASIC Monitor Calls
88 Decimal to Hex Conversion
88 Applesoft Sample Program
88 Step and Trace Peculiarities

PREFACE

The Apple II Reference Manual contains a complete assembly listing of the Monitor program in the Apple II. The Apple II Monitors Peeled Manual (this book) contains descriptions of the various routines in the Monitor and address tables arranged by topic instead of in the sequence of location within the machine. The material you find here has been chosen and organized to allow programmers of the Apple II to make convenient use of routines in the Monitor from their programs.

Many of the CALLable points in the Monitor fall under more than one topic. The layout of this book is intended to minimize the necessity of page flipping and cross referencing, so those points which seem to be appropriately described under more than one topic will be found in each applicable table.

This document covers the Apple II Monitor (both the Old Monitor and the Autostart Monitor versions), ROM address range $F8ØØ-$FFFF. This publication does not cover BASIC, APPLESOFT, DOS, HIRES, SWEET16, or Floating Point Arithmetic utility routines.

INTRODUCTION

There are two Monitor ROM's available for the Apple II. The two Monitors are identical for most functions. They differ only in certain features. This book describes both Monitors, with indications provided whenever the information applies to only one of the two.

Some thousands of Apple II computers have been shipped with the earlier version of the Monitor. In this book, that will be referred to as the Old Monitor. In 1979, a new version of the Apple II Monitor was developed. This Monitor contains new features to facilitate system start-up and program editing, at the expense of removing the instruction trace and single step facilities and sixteen bit multiply-divide routine of the Old Monitor. This new Monitor is called the Autostart Monitor in this book. The Autostart Monitor is available from Apple Computer Inc. and from many computer dealers under the name Autostart ROM, Apple Part No. A2MØØ27.

It is easy to determine which Monitor is in a machine. If the machine comes up with the APPLE II legend at the top of the screen when the power is turned on, the machine contains the Autostart Monitor. If the machine comes up with the Monitor prompt (*) then it contains the Old Monitor.

A program can also determine whether the Monitor is the Old or the Autostart ROM. The byte at $FAFF (64255 or -1281) contains $ØØ in the Autostart and $Ø1 in the Old Monitor.

OVERVIEW

CHAPTER 1

MONITOR USAGE MEMORY MAP
Use of memory by the Monitor and by the Apple II for machine control and display to the screen.

PAGE ZERO
Description in detail of all memory locations in page zero used by the Monitor, indicating legal range of values and all routines which use the location.

PAGES ONE THROUGH THREE
General descriptions of pages one and two and specific description of fields in page three.

PAGES FOUR THROUGH SEVEN AND ELEVEN
Description of how text is maintained in "screen refresh memory" for display on the screen, both primary and secondary display areas for text and Low Resolution (Color) graphics.

PERIPHERAL CONTROLLER WORK AREAS
A chart showing the scratchpad areas available in RAM memory for use by peripheral controller programs.

CHAPTER 2

KEYBOARD INPUT DIVISION OF LABOR
Descriptions of the lower level routines used by the Monitor to read data from the keyboard, including subroutines for cursor movement without reading characters.

USER CALLS TO KEYBOARD INPUT ROUTINES
Specifications for user calling of the routines at all levels for input of characters from the keyboard and for user program simulating (replacing) the keyboard as the input device.

KEYBOARD INPUT MONITOR ROUTINE
Table 1 contains addresses for character by character input from the keyboard via the routines described in the previous section. Table 2 contains addresses for line input from the keyboard.

OVERVIEW - TEXT OUTPUT TO THE SCREEN
Because there are so many ways to write text to the screen, this section contains an overview of the following pages on screen output.

TEXT OUTPUT WITHIN THE SCROLL WINDOW
Detailed description of the normal method of printing data to the screen, as used by PRINT of BASIC, including page zero reference table for Scroll Window services.

SCREEN FORMAT CONTROL BY ROUTINE
Table of addresses of routines in the Monitor which control the format of the Scroll Window and the format of data display.

SCREEN FORMAT CONTROL BY POKE/STORE
Description of methods of controlling the screen display format without calling routines in the Monitor.

SCROLL WINDOW DATA MANIPULATIONS
Table of routines which affect the data displayed in the Scroll Window, such as clearing part of it or scrolling it.

CURSOR POSITION CONTROL
Description of facilities for moving the cursor relative to current position or to an absolute location.

GENERAL TEXT TO THE SCREEN
Printing data to the screen whether some other device has been established (via CSWL) or not, and printing some things by a call to a Monitor routine which loads the A-reg and calls COUT itself.

TEXT OUTPUT WITHOUT THE SCROLL WINDOW
Ways and means of handling the screen as a formatted display device, with or without part of the screen being defined as a Scroll Window.

SECONDARY DISPLAY AREAS
Different methods of getting data into the secondary text display area.

CHAPTER 3

OVERVIEW OF INTERRUPT PROCESSING
General and specific definition of interrupts and interrupt processing with regard to computers in general and the Apple II in particular.

RESET INTERRUPT - OLD MONITOR
Description of handling a RESET interrupt with address table allowing user call to subsets.

RESET INTERRUPT - AUTOSTART MONITOR
Description of handling a RESET interrupt with address table allowing user call to subsets. Description of Soft Entry Vector setup and use.

IRQ/BRK INTERRUPT HANDLING
Descriptions of handling these types of interrupts by both Monitors, with Address Tables.

CHAPTER 4

MACHINE LANGUAGE DEVELOPMENT AIDS
Address table for routines in the Monitor which can be called to provide debugging information either by moving the information to some other place in memory or printing information through COUT.

LORES PLOTTING
Descriptions of the routines in the Monitor which support this function, with a table of addresses for directly calling them.

DATA MANIPULATION FUNCTIONS
Description of the routines in the Monitor which move data from one place to another, or change the format, or operate on one item with regard to another.

MONITOR COMMAND PROCESSOR
How to call the Monitor Command Processor, to have it execute Monitor commands and return to caller or stay in Monitor mode.

SPEAKER (BELL) USE THROUGH THE MONITOR
No music here. This is a description of how to use the speaker as a signaling device in the same manner as the error alarm or RESET key alarm.

CASSETTE TAPE INPUT AND OUTPUT
Description of all the routines involved with reading or writing of tape, with user call information specified for the high level routines. Includes list of calling programs for each point.

PADDLES, BUTTONS, AND ANNUNCIATOR I/O
Description of paddle reading for the machine language programmer and addresses to use for all these devices.

WAIT ROUTINE
This routine will take control of the machine for a length of time depending upon the input A-reg value. Table and formula are provided for use where interval between events is critical.

USE OF CONTROL-Y WITH PARAMETERS
Sample machine language program for rapid reading of the paddles.

REGISTERS FOR BASIC MONITOR CALLS
The Monitor GO command routine makes it possible to call from BASIC most Monitor routines which receive input in registers.

DECIMAL TO HEX CONVERSION
A sample program that shows how to convert from decimal to hexadecimal.

STEP AND TRACE PECULIARITIES
Differences between operation of the machine with and without Single Step in the Old Monitor.

CHAPTER 1

MEMORY ALLOCATION

MONITOR USAGE MEMORY MAP

Memory is divided into 256 byte sections, generally referred to as "pages". As with most countable items in computers, memory pages are numbered from zero. Page zero is very special in that the full address of a byte in page zero may be expressed in a single byte. Many 65Ø2 processor instructions are only two bytes in length because the operand is in page zero. Thus, Monitor usage of page zero receives heavy treatment in the following section.

Page one (address range $Ø1ØØ-$Ø1FF) is also special in the Apple II. This entire 256 byte area is called the "stack". The stack is a temporary storage area for which special instructions are provided in the 65Ø2. The contents of the A-register or P-register may be pushed onto the stack, which means the contents of the indicated register are stored in the stack at the location currently specified by the S-register: then the S-register is decremented. Data may be pulled or popped from the stack, which means that the S-register is incremented, and then the byte pointed at by the S-register is picked up into the appropriate register. A JSR instruction causes the current contents of the Program Counter to be pushed onto the stack before the jump. An RTS instruction pulls two bytes from the stack into the Program Counter.

The Monitor contains instructions which use the stack. However, the Monitor does not initialize the stack pointer register to a preset value or load the S-reg at any time.

Page two (address range $Ø2ØØ-$Ø2FF) is defined in the Apple II as the keyboard input area. The Monitor routines which support reading of the keyboard store the information into page two for use by the calling program after the next carriage return is detected.

Page three is address range $Ø3ØØ-$Ø3FF. Most of this area is unused by the Monitor. Quite often the first 2ØØ or so bytes are used for machine language programs called by APPLESOFT or BASIC programs. The Monitor uses only the last 16 bytes, as described in the Page Three Address Table. (Note, however, that DOS uses the 32 bytes before the Monitor's 16.)

Pages four through seven comprise the primary text or color graphics display area. Pages eight thru eleven comprise the secondary text or color graphics display area when that feature of the Apple II is used. However, page eight is generally the first page of the user area. In the address table, pages four thru seven and eight through eleven are described together when specifying memory address per screen line.

From address $Ø8ØØ to the end of memory in the machine is the user area for programs and data. However, if High Resolution Graphics is in use, then memory area from $2ØØØ through $3FFF is the primary display area for that function and $4ØØØ through $5FFF may be used as the secondary display area for that function.

RAM MEMORY ALLOCATION BY ADDRESS

```
ØØØØ |Page zero
ØØFF |
----------------------------------------------------------------
Ø1ØØ |Stack
Ø1FF |
----------------------------------------------------------------
Ø2ØØ |Keyboard Input
Ø2FF |
----------------------------------------------------------------
Ø3ØØ |Available
Ø3CF |
Ø3DØ |DOS
Ø3EF |
Ø3FØ |Vectors
Ø3FF |
----------------------------------------------------------------
Ø4ØØ |Primary Text
Ø7FF |and LORES Area
----------------------------------------------------------------
Ø8ØØ |User Program     |Secondary Text |RAM APPLESOFT
     |and Data space   |and LORES      |COMPILER/
ØBFF |to RAM size.     |               | INTERPRETER
     |                 |---------------|
     |ROM APPLESOFT    |               |
     |USER PROGRAM     |---------------|
2ØØØ |                 |Primary HIRES  |
     |INTEGER          |               |
2FFF |BASIC DATA       |               |
     |                 |               |----------------
3FFF |                 |               |RAM APPLESOFT
     |                 |---------------|USER PROGRAM
4ØØØ |                 |Secondary HIRES|
     |                 |               |
5FFF |                 |---------------|
     |
     |
3FFF -end 16K machine
     |
7FFF -end 32K machine
     |
BFFF -end 48K machine
```


PAGE ZERO

The Monitor makes use of the page zero locations from 32 ($2Ø) through 73 ($49) for general functions and normal operations. Locations 74-77 ($4A-4D) are not touched by the Monitor. Locations 78-79 ($4E-4F) are modified as described below to provide a random number starting point for an application program.

In addition, the Old Monitor uses locations 8Ø-85 ($5Ø-55) for the 16 bit Multiply and Divide routines (which are available for problem program use but are not used by any other part of the Monitor). These locations are not used by the Autostart Monitor.

The Autostart Monitor uses locations Ø and 1 during system initialization. This initialization is described in the section on "RESET Interrupt - Autostart Monitor" and below in describing the use of locations Ø and 1.

PAGE ZERO FIELDS

Dec Addr	Hex Addr	Monitor Label	Description
ØØ Ø1	$ØØ $Ø1	LOCØ LOC1	These locations are used by the Autostart Monitor during the automatic Disk Bootstrap function which takes place when the computer is powered up. Using these locations for indirect addressing, the slot addresses are checked - from slot 7 down thru slot 1 - to determine presence of a disk controller. If one is found, a Jump Indirect via $ØØ-Ø1 is executed to initiate the bootstrap operation.
32	$2Ø	WNDLFT	Left column of the Scroll Window: Range is Ø to 39 ($27). This field is used only in VTABZ which sets BASL,H to the memory location corresponding to CV and WNDLFT. The contents, when changed by user program, become effective on the next scroll operation, clear to end of page operation, or carriage return output. CH contains cursor horizontal position relative to (WNDLFT). After changing the contents of WNDLFT, either CALL VTAB or print a carriage return to the screen to make it take effect.
33	$21	WNDWDTH	Width of the Scroll Window: Range is 1 to 4Ø-(WNDLFT). When a character is written through COUT to the screen it is placed at (BASL),(CH), after which CH is incremented. Then (CH) is compared with (WNDWDTH) to determine whether the cursor has exceeded the right margin of the Scroll Window.

Dec Addr	Hex Addr	Monitor Label	Description
34	$22	WNDTOP	Top line of the Scroll Window: Range is Ø to 22 ($16) for full text screen. Range is 2Ø to 22 ($14 and $15) for mixed graphics and text. Valid values for VTAB in Basics are 21, 22, 23. This field is used during a scroll operation to indicate the line on which the operation should start. It is also the line on which the cursor is placed on completion of a HOME operation (clear the window, place cursor at top left).
35	$23	WNDBTM	Nominally, bottom line of Scroll Window: Range is (WNDTOP)+1 to 24 ($18). WNDBTM contains the number of the first line below the Scroll Window. Contents of WNDBTM are tested only on output of a carriage return ($8D) or line feed ($8A). It is used by Clear to End of Page and by Scroll routines.
36	$24	CH	Displacement from WNDLFT where next character to the screen will be placed: Range is Ø to (WNDWDTH) - 1. After the screen output routine STOADV places a character into the screen area as part of normal character output, CH is then incremented and compared to WNDWDTH. If CH is not less than WNDWDTH, a carriage return will be simulated. Note that CH is used for echoing keyboard input to the screen by the Monitor GETLN etc. routines.
37	$25	CV	Vertical screen position (line number) for next character to be written to the screen: Range is Ø to 23 ($17). The content of CV is relative to the top of the screen, not to the top of the Scroll Window. It may be set by loading the desired line number into A-reg and calling TABV. It may be set by POKEing the line number into CV and then calling VTAB. Actual storage of a character into the screen area includes use of BASL,H for line number, not CV. The calls above to VTAB or TABV are to set BASL,H from CV (and WNDLFT) for immediate future reference. If CV is at or below WNDBTM it will remain on the current screen line as carriage returns go by while the contents of the Scroll Window will be scrolled for each.

Dec Addr	Hex Addr	Monitor Label	Description
38 39	$26 $27	GBASL GBASH	Memory address within the screen area of the left end point of the desired line for LORES plot. This field is set by the GBASCALC routine to the memory location appropriate for the line number specified in the A-reg. See MASK at $2E.
4Ø 41	$28 $29	BASL BASH	This two byte field is the memory address for the left end character position of the current text line, line, within the Scroll Window. The contents are a function of CV and WNDLFT. This field is set by the BASCALC routine to point to the memory address for the left end of the screen line specified in the A-reg. This call to BASCALC is usually accomplished by the VTAB routine, which then adds (WNDLFT) to BASL,H to point to the left end of the line within the Scroll Window.
42 43	$2A $2B	BAS2L BAS2H	This two byte field is used as a work area only during a scroll operation. It is the destination line pointer used as each line is moved to the position above current.
44	$2C	H2	Right end point of horizontal line being drawn by the HLINE routine: Range is Ø to 39 ($27). This byte is set by the calling program before HLINE is called.
"	"	LMNEM	Low byte of two byte pointer (LMNEM, RMNEM) used by Disassembler as index to mnemonics table.
"	"	RTNL	Save area used by the Instruction Trace routine of the Old Monitor. Bottom point of a vertical line being drawn by VLINE routine: Range is Ø to 39 ($27) for mixed screen, Ø to 47 ($2F) for full screen graphics. This byte must be set before VLINE is called. Note that this byte is used when the Clear Screen (CLRSCR) routine uses VLINE to clear the screen.
"	"	RMNEM	Used with LMNEM as table index for mnemonic table by the Disassembler.
"	"	RTNH	Used with RTNL as a save area by the Instruction Trace routine of the Old Monitor.

Dec Addr	Hex Addr	Monitor Label	Description
46	$2E	MASK	With this label, this location is used as a $Ø F or $FØ by PLOT depending on whether the point is on the high side or the low side of the two horizontal plot lines represented by the GBASL,H pointer. Each location of the form (GBASL),Y contains two points on the screen, one above the other. MASK is used to set the appropriate one while leaving the other unchanged.
"	"	FORMAT	Using this label, the Disassembler uses this byte as temporary storage for the code which indicates the format of the instruction for display purposes.
"	"	CHKSUM	This byte is used during cassette tape read to continually accumulate the checksum which will be compared to that generated during the write operation which created the record. This byte is initialized to zero at the beginning of a tape read. As each byte is stored into memory it is Exclusively ORed against CHKSUM. After the last byte has been stored, one more byte is read from the tape and compared to CHKSUM. if equal, a good read may be assumed. As this result is not finally stored back into CHKSUM, that field cannot be used by the calling program to determine success or failure of the read. A method for this determination will be found in the section "Cassette Tape input and Output".
47	$2F	LASTIN	With this label, the RDBIT routine uses this byte as a work area to determine whether the sense of input from the cassette tape input register has changed.
"	"	LENGTH	This field is set by the Disassembler to indicate the length of the instruction. After output of the disassembled instruction, PCADJ uses this value to compute new values for PCL,H, which are returned to caller in the A and Y reg for user storage to PCL,H. Instruction trace in the Old Monitor also uses this field to indicate the number of bytes to move to the instruction trace execution work area (XQT).
"	"	SIGN	After a call to MULPM or DIVPM (signed 16 bit multiply or divide in the Old Monitor), the $Ø1 bit of this byte is set if the always-positive result is to be complemented by the calling program.

Dec Addr	Hex Addr	Monitor Label	Description
48	$3Ø	COLOR	This byte contains the code for the color of points to be placed on the screen in graphics mode. The SETCOL routine is entered with a value in the low order four bits of the A-reg. This value is then placed in both the high and low nibbles of COLOR. COLOR is then used with MASK in setting the value of the byte in the screen area to accomplish setting a particular point to the selected color. Color can be set directly by stuffing the value multiplied by $11 in color. For example, color = orange (9): From assembly - LDA #99, STA color. From BASIC - POKE 48, 9*17.
49	$31	MODE	This byte is used by the Monitor command processing routines to control parsing and to control operations when a blank is encountered after the hex digits. For example, a hex address followed by a colon causes setting of MODE so that during further processing of the input line each blank encountered signifies end of a hex value to be placed in memory. During parsing, the contents of MODE indicate where the hex values should be stored for use when the command itself is encountered. MODE is set to appropriate values by plus, minus, colon, and period.
5Ø	$32	INVFLG	This byte is a mask used by COUT1 to cause characters written to the screen area to display white on black (INVFLG=$FF) or black on white (INVFLG=$3F) or blinking (INVFLG=$7F). This field is set to $FF when a RESET occurs by the routine at SETNORM. The routine called SETINV can be called to set reverse video. The Monitor does not set blinking.
51	$33	PROMPT	This byte contains the prompt character which is written to the screen by the Monitor GETLN routine in preparation for reading a line of characters from the keyboard. When the RESET key is pressed, the Old Monitor quickly enters the MON routine, at which point the PROMPT field is set to $AA, "*". The Autostart Monitor also sets the "*" prompt character at the MON routine, but this is not necessarily a part of processing the RESET interrupt.

Dec Addr	Hex Addr	Monitor Label	Description
52	$34	YSAV	This byte is a save area used by the Monitor Command Processor. The Y-reg is used by the Command Processor in indexing through the input line. When a command has been decoded, the Y-reg is saved at YSAV before going to the selected service routine. On return to the Command Processor, the Y-reg is reloaded from here before transfer of control to NXTITM to continue scanning the input line.
53	$35	YSAV1	This byte is a save area for the Y-reg across a call to the screen output routines. Y-reg is saved and restored in the COUT1 routine.
54 55	$36 $37	CSWL CSWH	This two byte field contains the address of the routine which is to receive and dispose of output characters. When the RESET key is pressed this field is initialized to point to COUT1 to send output characters to the screen. Entering a Monitor Command nPc (n=port number, Pc=control-P) will cause the Monitor to set CSWL to ØØ, CSWH to Cn. The routine at that location will then receive (in the A-reg) each byte "written" through COUT, which is a JMP (CSWL). If the Monitor Command "ØPc" is executed, CSWL,H is set to point to COUT1 instead of to CØØØ.
56 57	$38 $39	KSWL KSWH	This two byte field contains the address of the user input routine. It is set by RESET key processing to point to KEYIN which gets its input from the keyboard. The Monitor Command nKc (n=port number, Kc=control-K) causes the setting of KSWL to ØØ, KSWH to Cn. This routine is then called any time the Monitor or executing program asks for another byte of input by calling RDKEY or one of the routines which in turn calls RDKEY. The Monitor Command "ØKc" results in setting KSWL,H to point to KEYIN instead of to CØØØ.
58 59	$3A $3B	PCL PCH	This field is a save and control area for the Program Counter. In addition to the Mini Assembler to keep track of where the next instruction is to be placed. When a BRK instruction is executed, this field is set to indicate the address stacked by the 65Ø2, pointing to two bytes beyond the BRK instruction executed.

Dec Addr	Hex Addr	Monitor Label	Description
			This field is used during Monitor commands L and G (Disassembler and Monitor "GOTO"). During disassembly of instructions this field is incremented as required. This field is used for a Jump indirect in execution of the Monitor G command. Updating of this field is accomplished with the assistance of the PCADJ routine whenever use requires incrementing in accordance with the length of the instructions. (See LENGTH at 47 or $2F.) On return from PCADJ, store A to PCL and Y to PCH to accomplish update. This field is used by the Old Monitor in support of Monitor commands S and T (single instruction step and instruction trace). For those functions, it is maintained as a pointer to the next instruction to be handled.
6Ø 61-67	$3C $3D-$43	XQT XQTNZ	This field is used as a work area for instruction step and trace in the Old Monitor. The field is eight bytes long and overlays A1L,H; A2L,H; A3L,H; and A4L,H. The next instruction to be executed (indicated by the contents of PCL,H) is moved to this field, possibly modified depending on instruction type, and then executed here. This field is not defined in the Autostart Monitor.
6Ø 61	$3C $3D	A1L A1H	Multipurpose Monitor work area: May be clobbered by Instruction Trace in the Old Monitor; see XQT above. When the Monitor begins processing a command, MODE is initialized to zero. As the input line is scanned, hex digits are first placed into A2L,H. From there they are moved also to A1L,H and A3L,H as long as MODE remains zero. When a plus, minus, colon, or period is encountered, MODE is modified to indicate which, and A1L,H will continue to contain the value, terminated by the operator encountered. A1L,H is the primary index for the BLANK Monitor command, memory examine or display. A1L,H contains the addend for the Monitor ADD command.

Dec Addr	Hex Addr	Monitor Label	Description
			A1L,H contains the minuend for the Monitor SUBTRACT command.
			A1L,H is the source field pointer during the Monitor MOVE command.
			A1L,H is one of the two indices used in the Monitor VERIFY command.
			A1L,H is the source field from which PCL,H is set on L and G Monitor commands, and the Old Monitor commands S and T, if an address is specified. If no address is used in the input line, then PCL,H is the residue of the last command which maintained or used it.
			A1L,H is the memory pointer used for cassette tape READ and WRITE Monitor operations.
			Monitor routine NXTA1 increments A1L,H by one and then compares the result to A2L,H. If A2L,H is less than A1L,H, then Carry is set when control is returned to the calling program.
62 63	$3E $3F	A2L A2H	Multipurpose Monitor work area: May be clobbered by Instruction Trace in the Old Monitor; see XQT above.
			This field is the receiving field into which hex data is stored from the input area during Monitor Command parsing. When the command itself is encountered, A2L,H contains the last parameter entered. While MODE contains zero (until a plus, minus, colon, or period is encountered) A2L,H is continually copied into A1L,H and A3L,H. If a "less than" sign is encountered, A2L,H is immediately copied to A4L,H and A5L,H.
			A2L,H is used to terminate examine (memory display), tape write, tape read, memory move, and memory verify operations.
			A2L,H contains the subtrahend in a Monitor SUBTRACT command operation.
			A2L,H contains the augend in a Monitor ADD command operation.
			A2L,H is the source field and A3L,H is maintained as the pointer for the Monitor STORE command.

Dec Addr	Hex Addr	Monitor Label	Description
			A2L,H contains the port number in an input port select or output port select (control K or P) command.
			Monitor routine NXTA1 increments A1L,H by one and then compares the result to A2L,H. If A2L,H is less than A1L,H then Carry is set when control is returned to the calling program.
64 65	$40 $41	A3L A3H	Multipurpose Monitor work area: May be clobbered by Instruction Trace in the Old Monitor; see XQT above.
			A1L,H and A3L,H are both filled from A2L,H during Monitor Command processing scan of the input line as described above regarding A1L,H.
			A3L,H is used as the destination pointer during Monitor STORE command processing.
			A3L,H is used as a work area by the Register Display routine, which is called by the control-E Monitor command, or as part of the single cycle or trace operations of the Old Monitor.
66 67	$42 $43	A4L A4H	Multipurpose Monitor work area: May be clobbered by Instruction Trace in the Old Monitor; see XQT above.
			This field (and A5L,H) are loaded from A2L,H during Monitor Command Processor scan of the input area when a "<" character is encountered.
			A4L,H is the receiving field pointer during a Monitor MOVE command execution.
			A4L,H is the second field pointer during a Monitor VERIFY operation.
			Monitor routine NXTA4 increments A4L,H by one, and then drops into NXTA1, which increments A1L,H by one and then compares the result to A2L,H. If A2L,H is less than A1L,H then Carry is set when control is returned to the calling program.

Dec Addr	Hex Addr	Monitor Label	Description
68	$44	A5L	Multipurpose Monitor work area:
69	$45	A5H	This field is not within the bounds of the area of XQT, which, in the Old Monitor, overlays A1L through A4H.

NOTE:

A5H = $45 = ACC — This field is filled from A2L,H as described above for A4L,H. However, the field is not otherwise referenced within the Monitor, except that ACC (below) is also A5H.

Dec Addr	Hex Addr	Monitor Label	Description
69	$45	ACC	This five byte field is a register save area.
7Ø	$46	XREG	With the following exceptions, the 65Ø2 registers
71	$47	YREG	are stored by the SAVE routine and reloaded by the
72	$48	STATUS	RESTORE routine.
73	$49	SPNT	

S-reg is stored at SPNT by SAVE but is never reloaded.

The A-reg is stored at ACC by the IRQ routine on either an IRQ interrupt or execution of a BRK instruction. On a BRK, entry into the SAVE routine at label SAV1 is used to store the rest of the registers. The other registers are not stored by the Monitor for an IRQ interrupt.

As described above, the registers are stored in this area on execution of a BRK instruction.

After execution of a BRK instruction or on execution of Monitor command control-E, the contents of this area are used to display the "registers" on the screen.

The registers (except S-reg) are loaded from this area before jumping to the requested location on execution of the Monitor C command.

In the Old Monitor Step and Trace command routines, the registers are stored here after each instruction execution and reloaded before the next traced instruction is executed.

Dec Addr	Hex Addr	Monitor Label	Description
74	$4A	unused	
75	$4B	unused	
76	$4C	unused	
77	$4D	unused	

Dec Addr	Hex Addr	Monitor Label	Description
78	$4E	RNDL	Random number field, 16 bits:
79	$4F	RNDH	This field is continually counted up by the KEYIN routine while testing for key pressed. Thus, the results are effectively random as it doesn't take long to overflow and start over. There is no other reference to this field within the Monitor.
8Ø	$5Ø	ACL	These three two-byte fields are used only by the
81	$51	ACH	multiply and divide routines in the Old Monitor.
82	$52	XTNDL	These routines are not called from any place in
83	$53	XTNDH	the Monitor. Therefore, these fields are used
84	$54	AUXL	only if a user program makes use of the multiply
85	$55	AUXH	or divide routines.

The section on Data Manipulation Functions contains a full description of the multiply and divide routines.

PAGES ONE THROUGH THREE

PAGE ONE ($Ø1ØØ-Ø1FF)

Page one is the hardware stack area. Monitor use of this area is only by means of the 65Ø2 instructions which use the stack, such as PHA, JSR, RTS, etc. The Monitor does not initialize or set the stack pointer (S-register) on a RESET or Power On interrupt or at any other time.

PAGE TWO ($Ø2ØØ-Ø2FF)

Page two is the Keyboard Input buffer area. At label "GETLN" the X-register is initialized as an index. At label ADDINP the character read from the keyboard is stored into page two indexed by the X-register. The result is that on return to the calling program the characters read from the keyboard have been stored in memory locations $Ø2ØØ and up, the last character stored being a carriage return, code $8D.

PAGE THREE ($Ø3ØØ-Ø3FF)

Page three contains "vectors" for special handling of certain interrupts at the high end of the page. The low end of the page, through $Ø3CF, is often used for machine language subroutines. From $Ø3DØ through $Ø3EF is used by DOS.

PAGE THREE ADDRESS TABLE

Hex	Dec	Function
$Ø3ØØ-$Ø3EF	768-1ØØ7	Not used by the Monitor.
$Ø3FØ-$Ø3F1	1ØØ8-1ØØ9	The Autostart Monitor uses this location as the BRK instruction interrupt vector (address).
$Ø3F2-$Ø3F3	1Ø1Ø-1Ø11	This is the RESET (Soft Entry) Vector (address) used by the Autostart Monitor, as described in the section "RESET Interrupt - Autostart Monitor".
$Ø3F4	1Ø12	Powered Up Indicator: if the Exclusive OR of "$A5" with the contents of $Ø3F3 is equal to the contents of $Ø3F4 then the RESET (Soft Entry) Vector is considered valid. Otherwise, a RESET interrupt will cause the Autostart Monitor to go through power-up initialization, including boot of DOS if available.
$Ø3F5-$Ø3F7	1Ø13-1Ø15	Reserved for APPLESOFT ("&" vector instruction).
$Ø3F8-$Ø3FA	1Ø16-1Ø18	Control-Y Vector (instruction).
$Ø3FB-$Ø3FD	1Ø19-1Ø21	Non-Maskable Interrupt Vector (instruction).
$Ø3FE-$Ø3FF	1Ø22-1Ø23	IRQ Interrupt Vector (address).

PAGES FOUR THROUGH SEVEN & ELEVEN

Address range $Ø4ØØ through $Ø7FF is the primary text and low resolution graphics display area. That is, screen display hardware displays on the screen the information stored in this part of memory.

Address $Ø8ØØ is generally the beginning of memory available to the user for general program or data storage. However, $Ø8ØØ through $ØBFF is the secondary text and low resolution graphics display area. By POKEing -16299 with any value, the screen display hardware can be directed to display to the screen from this secondary display area instead of the primary display area. POKE -163ØØ,Ø to switch back to the primary display area.

Although the hardware will display to the screen from the secondary display area, the Monitor does not support the feature. That is, the BASCALC and GBASCALC routines in the Monitor convert the line number input to the routine to the appropriate memory address for the primary display area only. Use of the secondary display area is described in the section "Secondary Display Areas".

Contiguous screen lines are not in contiguous memory locations. The characters on a screen line are in the same sequence in memory as on the screen, but the lines are mixed in a manner which simplifies the hardware display to the screen. The following table indicates for each line the address in memory for the leftmost character of the line in both the primary and secondary display areas.

The BASCALC routine in the Monitor computes the memory address for the line number input to that routine in the A-reg. Using the letters to designate bit positions in the input line number, the following indicates the result of the computataion:

```
Input line number (A-reg) ØØØABCDE
Memory address (BASH BASL) ØØØØØ1CD EABABØØØ
```

This can be arithmetically computed, using "modulo" arithmetic in place of the ANDs and ORs of machine language. For line number "L" (Ø- 23),

```
ADDR=1Ø24+256*((L/2) MOD 4)+(128*(L MOD 2))+4Ø*((L/8)MOD 4)
```

SCREEN MEMORY ADDRESS TABLE

Line	Primary Display Area		Secondary Display Area	
	Decimal	Hex	Decimal	Hex
Ø	1Ø24	Ø4ØØ	2Ø48	Ø8ØØ
1	1152	Ø48Ø	2176	Ø88Ø
2	128Ø	Ø5ØØ	23Ø4	Ø9ØØ
3	14Ø8	Ø58Ø	2432	Ø98Ø
4	1536	Ø6ØØ	256Ø	ØAØØ
5	1664	Ø68Ø	2688	ØA8Ø
6	1792	Ø7ØØ	2816	ØBØØ
7	192Ø	Ø78Ø	2944	ØB8Ø
8	1Ø64	Ø428	2Ø88	Ø828
9	1192	Ø4A8	2216	Ø8A8
1Ø	132Ø	Ø528	2344	Ø928
11	1448	Ø5A8	2472	Ø9A8
12	1576	Ø628	26ØØ	ØA28
13	17Ø4	Ø6A8	2728	ØAA8
14	1832	Ø728	2856	ØB28
15	196Ø	Ø7A8	2984	ØBA8
16	11Ø4	Ø45Ø	2128	Ø85Ø
17	1232	Ø4DØ	2256	Ø8DØ
18	136Ø	Ø55Ø	2384	Ø95Ø
19	1488	Ø5DØ	2512	Ø9DØ
2Ø	1616	Ø65Ø	264Ø	ØA5Ø
21	1744	Ø6DØ	2768	ØADØ
22	1872	Ø75Ø	2896	ØB5Ø
23	2ØØØ	Ø7DØ	3Ø24	ØBDØ

It is also interesting to note that although 24 lines of 4Ø characters computes to 96Ø bytes, the memory area described above contains 1Ø24 bytes per display area. The significance is that some of the bytes in

pages four through seven are not displayed on the screen. These bytes are eight groups of eight bytes each. This space has been set aside or allocated for use by peripheral controller cards in slots one through seven. The following table shows the allocation.

Misuse of these locations can be easily accomplished, with potentially serious results. Note that if an image of the screen is generated elsewhere and moved to this area in a block, the locations identified below will be modified. If a program is loaded from tape with the Monitor command mmmm.nnnnR, and if mmmm is less than $Ø4ØØ, then the bytes in the following table will be loaded from the tape. If an attempt is made to save the screen area to disk and later BLOAD it to the screen area, results can be confusing. The Disk Controller card, and possibly some peripheral device interface cards keep control information in these areas. For example, doing the above mentioned BLOAD from drive 2 when the BSAVE had been done from drive 1 will result in the disk switching back to drive 1.

The Reference Manual indicates that one must be sure that Scroll Window definition fields WNDLFT and WNDWDTH must not add up to more than 4Ø. Violation of the bytes in the following table will be the unfortunate result if this caution is not observed.

PERIPHERAL CONTROLLER WORK AREAS

Common (any/all) Decimal	Hex	Slot 1	Slot 2	Slot 3	Slot 4	Slot 5	Slot 6	Slot 7
1144	Ø478	Ø479	Ø47A	Ø47B	Ø47C	Ø47D	Ø47E	Ø47F
1272	Ø4F8	Ø4F9	Ø4FA	Ø4FB	Ø4FC	Ø4FD	Ø4FE	Ø4FF
14ØØ	Ø578	Ø579	Ø57A	Ø57B	Ø57C	Ø57D	Ø57E	Ø57F
1528	Ø5F8	Ø5F9	Ø5FA	Ø5FB	Ø5FC	Ø5FD	Ø5FE	Ø5FF
1656	Ø678	Ø679	Ø67A	Ø67B	Ø67C	Ø67D	Ø67E	Ø67F
1784	Ø6F8	Ø6F9	Ø6FA	Ø6FB	Ø6FC	Ø6FD	Ø6FE	Ø6FF
1912	Ø778	Ø779	Ø77A	Ø77B	Ø77C	Ø77D	Ø77E	Ø77F
2Ø4Ø*	Ø7F8*	Ø7F9	Ø7FA	Ø7FB	Ø7FC	Ø7FD	Ø7FE	Ø7FF

* Location 2Ø4Ø ($Ø7F8) has special significance. This location should be loaded with $CN, where N is the slot number of the active peripheral, whenever an interrupt may occur and the ROM/PROM expansion scheme is in use. This is necessary so that the return from interrupt software used allows the proper peripheral card to resume operation.

CHAPTER 2

INPUT AND OUTPUT

The default operation of the screen is as a scrolling device: new data is entered or output at the bottom of the screen and all above is shifted up line by line until the oldest information disappears off the top of the screen. With a little extra work in the user program, it is also possible to use the screen as a formatted display. Following is a description of the effects of that type of use, and some suggested solutions to the situations encountered.

Characters generated by the user program for display on the screen are handed to the Monitor one character at a time. The screen output handlers check for control character vs. display character, and operate in accordance with what they find. For example, output of a carriage return character or line feed character while the cursor is on the bottom line of the screen will cause a scroll operation to take place. If the screen is being used with a format instead of as a scroll device, then the program can easily avoid output of a carriage return or line feed when the cursor is on the bottom line of the screen.

The easiest way for the user program to read information from the keyboard is to call the Monitor at the point where it will read in a line (up to a carriage return) before returning control to the calling program. When this is done, the input information is always available at the same place in memory. There is, however, a conflict between using this type of a call and using the screen as a format type display. While the Monitor is receiving the keyboard input, it "echoes" the characters to the screen at the current cursor location. When end of input is signaled by a carriage return, the Monitor clears the cursor current line from cursor to the right end of the line (within the Scroll Window). Thus, the user program must make sure that before asking for input from the keyboard the cursor is placed where there is no significant data to the right.

It is possible to divide the screen into scroll area and non-scroll area. Many complications arise from this method of operation, so the recommended solution to the format display problem is to leave the screen full scroll and avoid scroll services when they are not desirable.

The entry points and qualifiers for using scroll and non-scroll areas will be found in the section on Text Output Without the Scroll Window.

KEYBOARD INPUT DIVISION OF LABOR

The Monitor routines supporting keyboard input are designed to echo the keyboard input to the screen (through COUT) at the current cursor position, and store the entered characters in the keyboard input area ($Ø2ØØ-$Ø2FF) for the convenience of the calling program. The executing program may position the cursor anywhere (in the Scroll Window) before calling the Monitor keyboard input routines. On entry of a carriage return from the keyboard, the Monitor keyboard input

routines will cause return of control back to the calling program with the character count plus one in the X-register and a carriage return in the input area as a terminator. The program need not look into the screen refresh memory to determine what was entered. (Note: The X-Register begins with a zero, so that if five characters are entered, the X-Register will reflect 4, although the actual value returned will be 5. X is incremented for the carriage return as well.

The routines described below are included in the address table. The following section, "User Program Calls ...", describes program setups for calling some of these entry points. Hex address, + Decimal address, and - Decimal address are given in brackets beneath each routine.

TABLE OF ROUTINES

Routine	Description
GETLNZ [$FD67] [64871] [- 665]	Entry at this point causes output of a carriage return (through COUT) before going to GETLN to write the prompt character and read the data.
GETLN [$FD6A] [64874] [- 662]	Entry at this point is with the cursor properly positioned (CV, BASL,H, and CH) as described in the section regarding Text Output Within the Scroll Window. GETLN prints the prompt character and initializes X-reg for indexed storage of the input characters into the input area. Control then goes to NXTCHAR.
NXTCHAR [$FD75] [64885] [- 651]	This is the top point in the character input loop. RDCHAR is called to get a character into the A-reg. On return the A-reg is tested for presence of the control-U (right arrow on the keyboard) and if it is found, the A-reg is then loaded from the screen refresh memory ((BASL),Y), assuming that the Y-reg contains the same value as CH. If the A-reg value is $EØ or greater, the lower case letter is converted to upper case by AND with $DF. The character is then stored from the A-reg to the input area. If the character is a carriage return, CLREOL is called to clear to blanks the rest of the window line, and then a conditional branch transfers control to COUT so that the RTS exit of COUT will return control to the calling program with the X-reg indicating the input character count +1. That is, the input is in memory locations $Ø2ØØ through $Ø2ØØ,X where $Ø2ØØ,X contains the carriage return. If the character is not a carriage return, then control is transferred to the NOTCR routine for display on the output device, and for interpretation with regards to control character affecting the input line.

Routine	Description
NOTCR [$FD3D] [64829] [- 7Ø7]	This routine receives control with the character of interest in (IN,X). The current setting of INVFLG is saved on the stack, while INVFLG is set to $FF so that the character "echoed" to the screen will be white on black. COUT is then called with the character in the A-reg. On return from COUT, INVFLG is restored from the stack. The character at IN,X is then tested for either of two special keys: Backspace (left arrow) or (line) Cancel (control-X). If Backspace, go to BCKSPC. If Cancel, go to CANCEL. If (IN,X) is neither Backspace nor Cancel the value of X-reg is tested to determine whether the input area is full or almost full. If there are more than 247 characters in the input area, a call to BELL is used to signal to the operator that the area is almost full.
NOTCR1 [$FD5F] [64863] [- 673]	After or without the margin warning bell, this routine gets control. Here, the X-reg is incremented to point at the next location in the input area to be filled. If, however, the result is overflow to zero, then entry of the Cancel key is simulated by falling into CANCEL. In the normal case, after incrementing the X-reg, control goes back to NXTCHAR to continue with character input and line building.
CANCEL [$FD62] [64866] [- 67Ø]	This routine prints a back-slash through COUT to indicate the action taken to the operator. Control is then passed to GETLNZ to initialize for entry of a new input line - the old one is gone.
BCKSPC [$FD71] [64881] [- 655]	On entry to this routine, the backspace character has already been printed through COUT with resulting backward movement of the cursor. If the current value in X-reg is zero, control is transferred back to GETLNZ for printing prompt and re-initializing for line input. Otherwise, the X-reg is decremented with control going to NXTCHAR to resume input of characters.
RDCHAR [$FD35] [64821] [- 715]	This routine calls RDKEY to get the next character placed into the A-reg. If, on return, it is found that the Escape key has been pressed, this routine calls the appropriate routine for reading the next character and performing the requested Escape key function. In the Old Monitor, control is passed to the ESC1 routine for this purpose, after a JSR to RDKEY to read the next character. In the Autostart Monitor, detection at RDCHAR of an Escape character transfers control (via ESC including RDKEY) to ESCNEW, which has the capability of handling multiple escape functions after a single depression of the Escape key. After any requested escape functions have been performed, control returns to RDCHAR as if there had been no interruption.

Routine	Description
RDKEY [$FDØC] [64780] [- 756]	This routine picks up and saves in the A-reg the character from the screen refresh memory area at BASL,H,CH (leaving the Y-reg filled with the contents of CH). It then changes that character in memory to blinking to indicate current cursor position. This routine asks for the next input character to be placed in the A-reg by doing an indirect jump via KSWL,H, which is normally pointing at KEYIN. Return is therefore to the caller of RDKEY, not to the RDKEY routine itself.
KEYIN [$FD1B] [64795] [- 741]	This is the routine which gets the next input key from the keyboard hardware. There are two required actions and two extra actions taken by this routine. The required actions are reading the keyboard input buffer over and over again until it is determined (by presence of the $8Ø bit) that a character has indeed been read. In this case, keyboard input buffer refers to the $1ØØ byte buffer at $2ØØ, and not to the location at $CØØØ. The sign flag is set or not by checking the status of the value at $CØØØ. If that value is positive, the routine loops back to KEYIN. If that value is negative, the value of $CØØØ is picked up and the keyboard strobe is referenced to prepare for the next keyboard input. The auxiliary actions taken by this routine are first, to count up the random number field, ignoring overflow, and second, to restore to the screen area the character modified by the RDKEY routine to remove the blink. This restore is accomplished by storing the A-reg at (BASL),Y, assuming that RDKEY loaded it. This is accomplished before the keyboard register is read into the A-reg. Return to the caller (of RDKEY) is accomplished by an RTS.
ESC [$FD2F] [64815] [- 721]	This routine is entered from RDCHAR if the A-reg is found to contain the Escape key code. It reloads the A-reg with a new key by calling RDKEY. In the Old Monitor, it then calls ESC1 to perform the requested single function. In the Autostart Monitor, ESCNEW is called to perform the requested functions. In either case, ESC is positioned such that the RTS which terminates Escape key processing returns control to RDCHAR.
ESCNEW [$FBA5] [64421] [-1115]	This routine exists only in the Autostart Monitor. It is the routine which supports cursor movement without data transfer; the Escape key functions I, J, K, and M. If the key next pressed is one of these four, the appropriate "old" function (Escape functions C, B, A, and D, respectively) is called. On return to ESCNEW, RDKEY is again called to get (and operate upon) the next character from the keyboard.

Routine	Description
	If the key pressed is not I, J, K, or M, then ESC1 is entered by JMP instead of JSR so that the RTS will return to the caller of ESCNEW instead of to ESCNEW.
ESC1 [$FC2C] [64556] [- 980]	In the Old Monitor this routine is called by the RDCHAR routine if the Escape key code is found in the A-reg by that routine. In the Autostart Monitor, control is passed in this case to the ESCNEW routine which then calls ESC1 or jumps to it depending on which key is pressed next. ESCNEW translates I, J, K, or M to C, B, A, or D respectively before calling ESC1, which returns to ESCNEW. If the key is other than I, J, K, or M, then ESCNEW JMP's to ESC1 with Carry set, to have the appropriate function performed. In this case, the next RTS will return control to the RDCHAR routine. When ESC1 is called, the contents of the A-reg (and the condition that Carry is "set") indicate the action to be taken. Control is transferred (conditional branch) to the appropriate Scroll Window Service routine to move the cursor without transferring data, or to clear all or some of the screen, or some combination of these.

CALLS TO KEYBOARD INPUT ROUTINES

The following paragraphs describe how to set up for calls to the various entry points in the Monitor for keyboard input, and what the results will be.

TABLE OF KEYBOARD INPUT CALLS

Routine	Description of Set-Up
GETLNZ	Write carriage return and prompt character, then read a line. Set-Up: X-reg, Y-reg, and A-reg are insignificant. CH is insignificant. CV should point to the line in the Scroll Window where input is to begin. BASL,H is insignificant. Results: CR is written, scroll takes place if appropriate. Prompt character is written through COUT. Keyboard is read character by character. Each character is placed at $Ø2ØØ,X and X is then incremented.

Routine	Description

Each character is "echoed" to the screen at cursor position and the cursor is then advanced.
On reading a carriage return, control is returned to calling program.

On Return:
- A-reg contains a carriage return code ($8D).
- X-reg contains the number of characters read before carriage return.
- Y-reg contains contents of WNDWDTH.
- Location $Ø2ØØ,X contains a carriage return.
- CH contains zero.
- CV contains line number, current value.
- BASL,H contains memory address for CV, WNDLFT.
- Window line is blank to the right of the end of the echoed input.

GETLN Write prompt character, then read a line.

Set-Up:
- X-reg, Y-reg, and A-reg are insignificant.
- CV and BASL,H should be compatible, pointing in the Scroll Window.
- CH indicates where on that line the prompt character is to be placed, to be followed by the echoed key input.
- Line address at which input is to begin must be in BASL,H. The Line number in CV will be calculated and set in BASL,H after a carriage return has been entered.

Results:
- Same as above for GETLNZ, with noted exception.

On Return:
- Same as above for GETLNZ.

NXTCHAR Enter here to bypass print of prompt character to the screen.

Set-Up:
- X-reg should be zero to begin storing input at $Ø2ØØ.
- Y-reg and A-reg are insignificant.
- CV and BASL,H should be compatible, pointing in the Window.
- CH indicates where echoing of keyboard input is to start.

Results:
- Same as above for GETLN.

On Return:
- Same as above for GETLNZ.

Note: For all the above, Escape key functions are supported as described in the reference material for the Monitor you have installed. Also, control-U (right arrow) is supported. When that character is recognized in the keyboard buffer, it is replaced in the A-register by the contents of the screen memory at the current cursor position.

Routine	Description

RDCHAR Read single character thru KSWL: return to caller in A-reg.

Set-Up:
- X-reg is insignificant, and will not be clobbered.
- Y-reg is insignificant.
- A-reg is insignificant.
- CV and BASL,H should be compatible, pointing in the Scroll Window to the line where input is to begin.
- CH indicates the horizontal position in the Scroll Window where cursor position will be indicated by blinking.

Results:
- The screen character at the cursor position (BASL),(CH) will be set to blinking until a key is pressed.
- If the Escape key is detected, the appropriate routines will be called to handle the requested function.
- Cursor right arrow (control-U) will be returned to the calling program, not the contents of the screen at the cursor.
- Cursor left arrow key (control-H) will be returned to the calling program.
- Characters read from the keyboard will not be stored in the $Ø2ØØ area.
- After the character is read, the blink will be turned off at the cursor position, but the key just read will not be echoed to the screen, nor will the cursor (CH) be advanced.
- Cancel input line (control-X) service is not defined as the data is not being stored in the $Ø2ØØ area.
- No special note is taken of carriage return, because the rest of the Monitor KEYIN Routine is not called. It is up to the calling program to take appropriate action on entry of a carriage return.

On Return:
- A-reg contains the value of the key pressed.
- Y-reg contains the contents of CH.
- X-reg is not affected by the routines called.
- CV, CH, BASL,H will have changed only if an Escape key function has been utilized.

RDKEY Read single character thru KSWL: return to caller in A-reg.

Set-Up:
- X-reg, Y-reg, and A-reg are insignificant.
- CV and BASL,H should be compatible, pointing in the Scroll Window.
- CH indicates the horizontal position where the cursor will be shown by blinking.

Routine	Description

Results:

The character on the screen at the cursor position is set to blinking.
KEYIN routine is given control via (KSWL) for physical reading of the keyboard.
Return (RTS) in KEYIN returns to the caller of RDKEY, not to the RDKEY routine.

On Return:

A-reg contains the character from the keyboard. It may be any character, including Escape, carriage return, right or left arrow, or any other control character.
X-reg is unchanged from the call.
Y-reg contains the contents of CH.
The character in the screen area at the cursor position has been restored to whatever it was before it was set to blink by RDKEY.
CV is used to calculate the new line.
BASL,H reflects the recalculated address.
CV remains unchanged.

KEYIN Read single character from keyboard: return to caller in A-reg.

Set-Up:

X-reg is unused and unaffected across this routine.
A-reg input to this routine is what will be stored into the screen area at the cursor position (BASL),Y to remove the blink condition after a key is pressed.
Y-reg is set to be used to store the A-reg into the screen area to remove the blink at (BASL),Y.
CH and CV are not referenced, but should be appropriately set. BASL,H are used as described for A-reg and Y-reg above.

Results:

On return to the caller, only the A-reg has been changed. It contains the input from the keyboard register.

KEYIN ROUTINE REPLACEMENT

There are cases in which it is desirable to replace the physical keyboard input routine with a routine which either reads from the keyboard and preprocesses the input, or gets the information to feed to the reading program from some source other than the keyboard. The requirements of such a program in replacing the KEYIN routine are described below. Placing the program/routine into effect is accomplished by storing the entry point in KSWL,H.

The replacement routine should manage the following resources as indicated.

A-reg Store the A-reg at (BASL),Y, then load the A-reg from whatever source is to be used.

X-reg Must be unaltered. Save on entry and restore on exit if it must be used by the replacement routine.

Y-reg Use as indicated above for A-reg.

It must not be changed on return from contents on entry, so save and restore if it must be used otherwise. (This caution is not required, however, if the source of the input prevents Escape key and right arrow from being entered. In such case, the Y-reg is expendable.)

CH
CV
BASL,H These are all used for echoing the "keyboard" input, so the replacement routine should either leave them alone or manipulate them in an appropriate manner.

NOTE: On replacing the pointer to KEYIN at KSWL,H, it is generally safer to pick up and store the current contents of KSWL,H in a save area before placing the address of your routine, and then restore KSWL,H from that save area when taking the replacement routine out of service.

NOTE: If you replace the contents of KSWL,H with the address of your routine while using DOS, expect the unexpected. DOS uses both CSWL,H and KSWL,H, and periodically restores them to appear the way DOS likes to see them regardless of current contents. Depending upon your application, it may be a good idea to replace both pointers on a temporary basis so that echo to the screen will not pass through DOS. But remember to repair both as soon as possible.

KEYBOARD INPUT MONITOR ROUTINE

There are many points in Keyboard Service which a user program could usefully call. However, because they are generally different locations in a continuous string of instructions, and all instructions after the point of entry will be used, sections of this table of addresses are in Monitor sequence rather than in sequence by potential usability.

Note that once the Monitor is jumped to at the specified point, all of the initialization described after that entry point is also performed. This is implied by the & at the end of each function description.

ADDRESS TABLE 1—CHARACTER INPUT

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
BOTH MONITORS					
Call RDKEY to get next character into A-reg. Compare to $9B (Escape). If = BR to ESC to call for next character and do Escape function. Else, RTS.	FD35	64821	-715	RDCHAR	A,Y
Set screen to blink at cursor saving original character in the A-reg from (BASL),Y &	FD∅C	6478∅	-756	RDKEY	A,Y
Jump Indirect (KSWL) to KEYIN	FD18	64792	-744		A
Increment random number at RNDL,H while polling keyboard register.	FD1B	64795	-741	KEYIN	A
Store A-reg to (BASL),Y (clear blink set by RDKEY routine). &	FD26	648∅6	-73∅		
Load A-reg from keyboard register and clear keyboard strobe and RTS.	FD28	648∅8	-728		A
Using character in A-reg, with Carry set, BR to routine for Escape key service. @ HOME clear scroll window A ADVANCE cursor right B BS cursor left C LF cursor down one line D UP cursor up one line E CLREOL clear to end of line F CLREOP clr to end of window other ignore: RTS	FC2C	64556	-98∅	ESC1	A,Y
Set port ∅ (keyboard) for input.	FE89	65161	-375	SETKBD	A,X,Y
OLD MONITOR ONLY					
Call RDKEY for Escape key service &	FD2F	64815	-721	ESC	A,Y
Call ESC1 with character in A-reg and Carry set to do indicated function. Return is to RDCHAR.	FD32	64818	-718		A,Y
AUTOSTART MONITOR ONLY					
Call RDKEY for Escape key service &	FD2F	64815	-721	ESC	A,Y
Call ESCNEW with character in A-reg and Carry set to do indicated function. Return from Escape processing is to RDCHAR (above).	FD32	64818	-718		A,Y

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Set Carry flag and JMP to ESC1 to handle Escape key functions A, B, C, D, E, F.	FB97	644∅7	-1129	ESCOLD	A,Y
Handle Escape key functions I, J, K, M. Translate to D, B, A, C and call ESCOLD. Then RDKEY to get next character and drop into ESCNEW to continue Escape key processing.	FB9B	64411	-1125	ESCNOW	A,Y
Escape key processing entry point. If A-reg contains I, J, K, or M then go to ESCNOW to translate and handle it with return to ESCNEW. Otherwise go to ESCOLD to handle this entry and exit from Escape mode.	FBA5	64421	-1115	ESCNEW	A,Y

BASL,H	4∅-41	$28-$29
KSWL,H	56-57	$38-$39

ADDRESS TABLE 2—LINE INPUT

Logically speaking, the place to start below is GETLNZ, but the sequence of presentation here is the sequence of instructions in the Monitor because of heavy use of "fall into" next code segment.

Note that once the Monitor is jumped to at the specified point, all of the initialization described after that entry point is also performed. This is implied by the & at the end of each function description.

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Echo keyboard input thru COUT to the screen, from IN,X, with INVFLG temporarily set to $FF. &	FD3D	64829	-7∅7	NOTCR	A
Pick up character from IN,X; if $88 goto BCKSPC. if $98 goto CANCEL. If X-reg (input index) greater than $F7 fall into FD5C. Else goto NOTCR1, bypass Bell.	FD4D	64845	-691		A
Sound bell if X indicates 248+ input characters. &	FD5C	6486∅	-676		
Increment X-reg; If X not zero goto NXTCHAR. If X=∅ fall into CANCEL.	FD5F	64863	-673	NOTCR1	X

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Load $DC (\) into A-reg.	FD62	64866	-67Ø	CANCEL	A,X,Y
Backward slash indicates line input cancelled.	&				
Call COUT to print A-reg.	FD64	64868	-668		
Then fall into GETLNZ.	&				
Print carriage return thru COUT.	& FD67	64871	-665	GETLNZ	A,X,Y
Load PROMPT into A-reg.	& FD6A	64874	-662	GETLN	A,X,Y
Call COUT to print A-reg.	& FD6C	64876	-66Ø		
Load X-reg with $Ø1 for passage thru backspace operation.	FD6F	64879	-657		A,X
If X=Ø goto GETLNZ to start over. Else, decrement X-reg and fall into NXTCHAR.	FD71	64881	-655	BCKSPC	A,X,Y
Call RDCHAR to get next character.	FD75	64885	-651	NXTCHAR	A
If character received is ctrl-U ($95, right arrow) pick up the screen character from (BASL),Y to replace it in the A-reg.	&				
If A-reg greater than $DF, then	FD7E	64894	-642	CAPTST	?A
AND against $DF to make it upper case.	&				
Store A-reg to input area at IN,X. Compare to carriage return. Goto NOTCR (above) if not. Else, call CLREOL to clear the rest of the line, then print carriage return thru COUT, using RTS from that function to accomplish return to caller of keyboard input.	FD84	649ØØ	-636	ADDINP	

IN =$Ø2ØØ, keyboard input area.
INVFLG is at $32 (5Ø).

OVERVIEW—TEXT OUTPUT TO THE SCREEN

The highest level of support in the Monitor for text output to the screen is scroll device support. In addition, the Monitor contains many components which support use of the screen in a formatted manner. Because there are so many ways to write text to the screen, the topic of screen output has been divided into the following sections:

TEXT OUTPUT WITHIN THE SCROLL WINDOW
describes the normal manner of text output, defining the fields in page zero which are used to control this function, and which are used in the descriptions in the following sections.

SCREEN FORMAT CONTROL
identifies the entry points by means of which display operation (full text, full graphics, mixed LORES graphics and text), Scroll Window setup, and character display mode (black on white or white on black or blinking) are established or modified.

SCROLL WINDOW DATA MANIPULATIONS
describes Monitor calls which clear all or part of the Scroll Window, set parts of the window to some user specified value, or cause conditional or unconditional scrolling of the window.

CURSOR POSITION CONTROL
describes the ways and means of moving the cursor relative to its current position, or moving it to some location independent of its current position.

GENERAL TEXT TO THE SCREEN
describes the Monitor entry points to output user program generated data to the screen or to the current output device if CSWL has been modified. Also, entry points are described to transmit standard types of output (blanks, bell code, carriage return) to the output device (generally screen).

TEXT OUTPUT WITHOUT THE SCROLL WINDOW
describes the entry points used for placing characters on the screen outside of the Scroll Window, and for reading the keyboard when echo to the Scroll Window is to be performed.

SECONDARY DISPLAY AREAS
describes various ways of using the Secondary Text area, even for limited Scroll Window functions such as allowing keyboard input echo to go to the Secondary area.

Any entry point which fits into more than one category will be found in each appropriate address table.

OUTPUT WITHIN THE SCROLL WINDOW

Scroll Window operation is compatible with printer or typewriter output in that new characters are displayed to the right of previous output, and new lines are displayed below previous lines. It is this mode of operation which is described in this section. That is, this section describes "printing" information by means of the CSWL vector to the screen or to a printer type device. The section on General Text to the Screen describes use of the screen, bypassing the CSWL vector and making direct use of the Scroll Window output routines.

The normal method provided in the Apple II for displaying output information is by "calling" COUT with the character in the A-reg for each displayable character or format control character (such as a carriage return). At COUT, a JuMP Indirect is done via the CSWL vector to the routine which will place the character on the selected medium

or accomplish the indicated control function. When the system is initialized, this vector is set to point to COUT1 which supports Scroll Window output to the screen. If the user sets a different output device (by PR#n in BASIC or ctrl-P in Monitor mode), then the CSWL vector will be set to pass the output bytes to the selected peripheral controller card instead of to the screen. Depending on which peripheral controller card, and which controls are active, the program on that card may place the character on the output device, and then JuMP to COUT1 to write it also to the Scroll Window.

The normal mode of text output to the screen is in "scroll" mode. In this mode, new information is written to the bottom line of the screen, and the contents of the screen are moved up, up, and away as required to allow entry of new information below the old. This mode of output is used in APPLESOFT or BASIC "PRINT" statements. This is the mode of output used by any Monitor command which displays data to the screen.

As new characters are written to the screen, they are placed at the position of the cursor. The cursor position is a location on the screen (and in screen refresh memory) specified by the contents of certain fields in page zero. Also, the Scroll Window is a portion (or all) of the screen as defined by the contents of certain fields in page zero. There is no special display hardware involved with the scrolling function. Routines in the Monitor move data in the screen refresh memory as required to support the scrolling function.

The fields in page zero describing the Scroll Window indicate the left column and width, and the top and bottom lines, as described here.

The cursor position is defined in various fields, and unless a user program interferes they will be compatible.

The screen line number of cursor position is contained in the field CV. CV indicates the line number of the cursor relative to the top line of the screen, not the Scroll Window. (Note that this is different from CH, described below.) The screen refresh memory location which corresponds to this line number is maintained in the two byte field (BASL,H). Note, however, that if the left edge of the Scroll Window is not the leftmost character of the screen, BASL,H will have been adjusted to point to the leftmost character position on that line within the Scroll Window. Thus, a program may interrogate CV to determine the line number of the cursor, but the program cannot just POKE a different line number into CV to move the cursor as BASL,H must be updated as well.

The horizontal position of the cursor is maintained in CH. The value in CH is relative to the left edge of the Scroll Window, not necessarily to the screen. When a character is being "written" or "printed" to the screen, the routine which places the character in screen refresh memory uses the Y-reg for horizontal position, in the assumption that it has been loaded from CH. In the address table, each description indicates whether the routine being called uses CH or the Y-reg.

For machine language programs, Scroll Window output is most easily accomplished by a JSR to COUT at $FDED (-531) with the byte in the A-reg. From BASIC the same thing is accomplished by PRINTing a variable in which the byte has been stored. In BASIC, of course, a whole string can be written with a single command.

As the characters are passed through COUT1, they are modified, if necessary, to be written in white on black, black on white, or flashing, in accordance with the contents of the field called INVFLG. This field can be set (POKEd) at any time, and is immediately effective on all future characters printed by the program until it is again modified. This function only applies to program print output. During keyboard entry, INVFLG is temporarily changed to $FF as each input character is echoed through COUT.

The two byte field BAS2L,H is described below although it is rather useless for user program reference. It is a work area used only during a scroll operation.

PAGE ZERO FIELDS

Dec	Hex	Routine	Description
32	$2Ø	WNDLFT	Left column of the Scroll Window: Range is Ø to 39 ($27). This field is used only in VTABZ. The contents, when changed by user program, become effective on the next scroll operation, clear to end of page operation, or carriage return output. CH contains cursor horizontal position relative to (WNDLFT). After changing the contents of WNDLFT, either CALL VTAB or output a carriage return to make it take effect.
33	$21	WNDWDTH	Width of the Scroll Window: Range is 1 to 4Ø-(WNDLFT). When a character is written through COUT to the screen it is placed at (BASL),(CH), after which CH is incremented. At that time (CH) is compared with (WNDWDTH) to determine whether the cursor has exceeded the right margin of the Scroll Window.
34	$22	WNDTOP	Top line of the Scroll Window: Range is Ø to 22 ($16) for full text screen. Range is 2Ø to 22 ($14 to $16) for mixed graphics and text. This field is used during a scroll operation to indicate where the operation should start.

Dec	Hex	Routine	Description
35	$23	WNDBTM	Bottom line of Scroll Window +1: Range is (WNDTOP)+1 to 24 ($18). WNDBTM indicates the first line number below the window. Contents of WNDBTM are tested only on output of a carriage return ($8D) or line feed ($8A). It is used by Clear to End of Page and by Scroll routines.
36	$24	CH	Displacement from WNDLFT where next character to the screen will be placed: Range is Ø to (WNDWDTH)-1. After the screen output routine STOADV places a character into the screen area as part of normal character output, CH is then incremented and compared to WNDWDTH. If CH is not low then a carriage return will be simulated. Note that CH is used for echoing keyboard input to the screen by the Monitor routines GETLN etc., because COUT is used.
37	$25	CV	Vertical screen position (line number) for next character to be written to the screen: Range is Ø to 23 ($17). The content of CV is relative to the top of the screen, not to the top of the Scroll Window. It may be set by loading the desired line number into A-reg and calling TABV. It may be set by POKEing the line number into CV and then calling VTAB. Actual storage of a character into the screen area includes use of BASL,H for line number, not CV. The calls above to VTAB or TABV are to set BASL,H from CV for immediate future reference. If CV is at or below WNDBTM, it will remain on current line as carriage returns go by while the contents of the Scroll Window will be scrolled for each.
4Ø 41	$28 $29	BASL BASH	This two byte field is the memory address for the left end character position of the current text line, within the Scroll Window. The contents are a function of CV and WNDLFT. This field is set by the BASCALC routine to point to the memory address for the left end of the line specified in the A-reg. This call to BASCALC is usually accomplished by the VTAB routine, which then adds (WNDLFT) to BASL,H to point to the left end of the line within the window.

Dec	Hex	Routine	Description
42 43	$2A $2B	BAS2L BAS2H	This two byte field is used as a work area only during a scroll operation. It is the destination line pointer used as each line is moved to the position above current.
5Ø	$32	INVFLG	This byte is a mask used by COUT1 to cause characters written to the screen area to display white on black (INVFLG=$FF) or black on white (INVFLG=$3F) or flashing (INVFLG=$7F). This field is set to $FF when a RESET occurs by the routine at SETNORM. The routine called SETINV can be called to set reverse video. The Monitor does not set flashing. Note: INVFLG=$7F does not cause all characters to flash: the upper 2 bits of the character must be Ø1 for flashing to occur.
53	$35	YSAV1	This byte is a save area for the Y-reg across a call to the screen output routines. Y-reg is saved and restored in the COUT1 routine.
54 55	$36 $37	CSWL CSHW	This two byte field contains the address of the routine which is to receive and dispose of output characters. When the RESET key is pressed, this field is initialized to point to COUT1 to send output characters to the screen. Entering a Monitor Command nPc (n=port number, Pc=control-P) will cause the Monitor to set CSWL,H to CnØØ. The routine at that location will then receive (in the A-reg) each byte "written" through COUT, which is a JMP (CSWL). If the Monitor Command "ØPc" is executed, CSWL,H is set to point to COUT1 instead of to CØØØ.

SCROLL WINDOW OUTPUT ROUTINES

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Jump via CSWL, character print.	FDED	65ØØ5	-531	COUT	none
Write byte in A-reg to screen at cursor (CV),(CH) using INVFLG and supporting cursor move.	FDFØ	65ØØ8	-528	COUT1	none
Write byte in A-reg to screen at (CV),(CH) with cursor move but not INVFLG.	FDF6	65Ø14	-522	COUTZ	none

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Print carriage return thru COUT.	FD8E	64910	-626	CROUT	A
Print thru COUT "ERR" and bell code.	FF2D	65325	-211	PRERR	A
Print bell code ($87) thru COUT.	FF3A	65338	-198	BELL	A
Set BASL,H from CV (and WNDLFT).	FC22	64546	-990	VTAB	A
Set BASL,H from (A) and WNDLFT without regard to CV.	FC24	64548	-988	VTABZ	A
Set BASL,H to left end of screen line (not window line) in A-reg.	FBC1	64449	-1087	BASCALC	A

CH	36	$24	WNDLFT	32	$20
CV	37	$25	WNDWDTH	33	$21
CBASL,H	38-39	$26-27	WNDTOP	34	$22
BASL,H	40-41	$28-29	WNDBTM	35	$23
INVFLG	50	$32			

SCREEN FORMAT CONTROL BY ROUTINE

This table identifies the places in the Monitor which control the display mode of operation and the Scroll Window configuration.

Function		Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Clear HIRES graphics mode.	&	FB33	64307	-1229		A
Set display area primary.	&	FB36	64310	-1226		A
Set TEXT mode.	&	FB39	64313	-1223	SETTXT	A
Load 0 into A-reg for WNDTOP, branch to SETWND below.	&	FB3C	64316	-1220		A
Set Graphics mode.	&	FB40	64320	-1216	SETGR	A,Y
Set mixed graphics/text mode.	&	FB43	64323	-1213		A,Y
Call CLRTOP to clear graphics.	&	FB46	64326	-1210		A,Y
Load 20 ($14) into A-reg for set of WNDTOP. Fall into SETWND.	&	FB49	64329	-1207		A
Set top line of window (WNDTOP) from A-reg, 0 or 20 or user set. Fall thru following.		FB4B	64331	-1205	SETWND	A
Load A-reg with 0 for WNDLFT.	&	FB4D	64333	-1203		A
Store A-reg to WNDLFT.	&	FB4F	64335	-1201		A
Load A-reg with 40 for WNDWDTH.	&	FB51	64337	-1199		A
Store A-reg to WNDWDTH.	&	FB53	64339	-1197		A
Load A-reg with 24 for WNDBTM.	&	FB55	64341	-1195		A
Store A-reg to WNDBTM.	&	FB57	64343	-1193		A
Load A-reg with 23 for VTAB.	&	FB59	64345	-1191		A
Store A-reg to CV.	&	FB5B	64347	-1189	TABV	A
Jump to VTAB - set BASL,H RTS.						

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Load Y-reg with $FF for INVFLG. Fall into SETIFLG.	FE84	65156	-380	SETNORM	Y
Load Y-reg with $3F for INVFLG. BR to SETIFLG.	FE80	65152	-384	SETINV	Y
Store Y-reg in INVFLG and RTS. $FF white on black (from SETNORM) $3F black on white (from SETINV) $7F flashing (characters from user call with upper 2 bits of 01)	FE86	65158	-378	SETIFLG	none
Set CSWL,H to point to COUT1.	FE93	65171	-365	SETVID	A,X,Y

CH	36	$24	WNDLFT	32	$20
CV	37	$25	WNDWDTH	33	$21
INVFLG	50	$32	WNDTOP	34	$22
BASL,H	40-41	$28-29	WNDBTM	35	$23
CSWL,H	54-55	$36-37			

SCREEN FORMAT CONTROL BY POKE/STORE

In many cases, the routine in the Monitor described on the previous page exists because the Monitor itself uses the function described. Often, calling the Monitor for a specific control function is doing it the hard way. This table indicates other ways of accomplishing the same results.

Function	Method
Set GRAPHICS display mode.	POKE -16304,0 or STA C050
Set TEXT display mode.	POKE -16303,0 or STA C051
Set GRAPHICS mode to full screen.	POKE -16302,0 or STA C052
Set MIXED GRAPHICS and TEXT mode.	POKE -16301,0 or STA C053
Set display to Primary Area.	POKE -16300,0 or STA C054
Set display to Secondary Area.	POKE -16299,0 or STA C055
Clear HIRES/Set LORES for graphics.	POKE -16298,0 or STA C056
Set HIRES Graphics mode.	POKE -16297,0 or STA C057
Set top line of Scroll Window.	POKE 34,line-number (0-23) Bottom must be greater than top.
Set left edge of Scroll Window.	POKE 32,column-number (0-39) Left edge + width not to exceed 40.
Set width of Scroll Window.	POKE 33,number-of-columns (1-40), Left edge + width not to exceed 40.
Set bottom line of Scroll Window.	POKE 35,line-number (1-24) Bottom must be greater than top.
Set Normal (white on black) text.	POKE 50,255 or store $FF in $32
Set Flashing text.	POKE 50,127 or store $7F in $32
Set Inverse (black on white) text.	POKE 50,63 or store $3F in $32

If the above means are used to change the Scroll Window configuration, the user program should also take steps to insure that the cursor has a valid position within the window (CV, CH, BASL,H). CALL -936 will place the cursor in the Window.

$CØ5Ø and $CØ51 control Text mode vs. all or some graphics. The other items regarding HIRES or LORES or full or part screen graphics may be established first, but will not be apparent until $CØ5Ø is tickled. Likewise, $CØ51 will bring back Text Mode regardless of the other settings.

SCROLL WINDOW DATA MANIPULATIONS

This table describes three types of Scroll Window data manipulation entry points. The first is Monitor Label ESC1, the Escape Key Processor, because it transfers control to a number of the other entry points depending upon the A-reg contents and Carry being set. One entry point of the Autostart Monitor is included because it handles one requirement of ESC1 - that Carry be set.

The second part of the table is a list of entry points supporting clearing or setting parts of the screen to a particular value.

The third part of the table describes points causing conditional or unconditional scrolling of the window.

ADDRESS TABLE

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Call screen data manipulation. If Carry is set and A-reg = @ goto HOME A goto ADVANCE B goto BS C goto LF D goto UP E goto CLREOL F goto CLREOP other RTS to caller. The RTS at the end of each of these functions returns control to the caller of ESC1.	FC2C	64556	-98Ø	ESC1	A,Y
Clear from line (CV) col (CH) to end of Scroll Window.	FC42	64578	-958	CLREOP	A,Y
Clear from line (CV) col (Y) to end of Scroll Window.	FC44	6458Ø	-956		A,Y
Clear from line (A) col (Y) to end of Scroll Window.	FC46	64582	-954	CLEOP1	A,Y
Clear Scroll Window to blanks, set cursor to top left corner of the window.	FC58	646ØØ	-936	HOME	A,Y
Set CH=Ø, CV=(A), clear to EOP (end of page = end of window).	FC5A	646Ø2	-934		A,Y
Clear window from line (A) to blank, set cursor to left end of line (CV).	FC5C	646Ø4	-932		A,Y
Clear line from cursor ((BASL),(CH)).	FC9C	64668	-868	CLREOL	A,Y
Clear line from cursor (BASL),Y.	FC9E	6467Ø	-866	CLEOLZ	A,Y
Set character in A-reg from cursor (BASL),Y to EOLine.	FCAØ	64672	-864	CLEOL2	A,Y
Clear line (BASL), then set BASL,H from CV and WNDLFT.	FC95	64661	-875	SCRL3	A,Y
Clear line from cursor (BASL),Y, then set BASL,H from CV & WNDLFT. CH remains unchanged.	FC97	64663	-873		A,Y
Zero to A-reg for CH. &	FC62	6461Ø	-926	CR	A,?Y
Store A-reg to CH. &	FC64	64612	-924		A,?Y
Increment CV. &	FC66	64614	-922	LF	A,?Y
Compare CV to WNDBTM. Set BASL,H; if (CV) < (WNDBTM), do scroll if required.	FC68	64616	-92Ø		A,?Y
Scroll the window, lines (CV) thru (WNDBTM).	FC7Ø	64624	-912	SCROLL	A,Y
Scroll the window, lines (A) through (WNDBTM).	FC72	64626	-91Ø		A,Y
Autostart Monitor extended service					
Set Carry flag and JMP to ESC1 to handle Escape key functions A, B, C, D, E, F.	FB97	644Ø7	-1129	ESCOLD	A,Y

CH	36	$24	WNDLFT	32	$2Ø
CV	37	$25	WNDWDTH	33	$21
INVFLG	5Ø	$32	WNDTOP	34	$22
BASL,H	4Ø-41	$28-29	WNDBTM	35	$23

CURSOR POSITION CONTROL

In general, the Cursor is at the position indicated by the contents of CV (line number relative to top of screen) and CH (column number relative to to the left margin of the Scroll Window). The memory location of the cursor is the sum of the contents of BASL,H (which contains the address of the leftmost character of the line within the Scroll Window) and the contents of CH. Normally, then, BASL,H contains an address computed from the contents of CV and WNDLFT. However, if either CV or WNDLFT is changed without recomputing BASL,H then the different routines of the Monitor may come up with unpredictable (or at least undesired) results.

In the following table, the description includes indication of which of the cursor address fields is being used for what. Note, for example, that at $FC95 the line indicated by BASL,H is cleared, and then BASL,H is recomputed from CV, WNDLFT for future references.

The ESC1 and VIDOUT routines are included in the table because they can be made to use (goto) the other entry points by passing them the appropriate A-reg contents on entry. VIDOUT is the routine which handles CR, backspace, and line feed when such characters are sent through COUT1 (generally thru COUT). ESC1 is the routine called to accomplish the desired function when the keyboard routines are operating in ESCAPE key mode. Thus, it has four way cursor movement capability, as well as the capability of clearing the Scroll Window from cursor present position to end of current line or end of the Scroll Window, or of clearing the entire Scroll Window and placing the cursor at the top left corner of it. The function performed depends upon the contents of the A-reg at entry, and the "set" condition of the Carry processor status bit.

Entry point ESCOLD of the Autostart Monitor is included in the table due to its relationship to ESC1.

The next group of points contains those which clear data on the screen as well as move the cursor.

The third group is entry points supporting movement of the cursor relative to its current position.

The fourth group supports positioning the cursor at a desired location without reference to its current position. To do this, the program should set CV and CH and then call VTAB to set BASL,H.

ADDRESS TABLE

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Call screen/cursor manipulation. If Carry is set and A-reg = @ goto HOME A goto ADVANCE B goto BS C goto LF D goto UP E goto CLREOL F goto CLREOP other RTS to caller. The RTS at the end of each of these functions returns control to the caller of ESC1.	FC2C	64556	-98Ø	ESC1	A,Y
Set Carry flag and JMP to ESC1 to handle Escape key functions A, B, C, D, E, F.	FB97	644Ø7	-1129	ESCOLD Autostart only	A,Y

Place character in screen memory or process control character.
If (A) > $9F or < $8Ø goto STOADV.
If (A) = $8D goto CR.
If (A) = $8A goto LF.
If (A) = $88 goto BS.
If (A) = $87 sound "bell".
If (A) = other ignore it; RTS
	FBFD	645Ø9	-1Ø27	VIDOUT	A,Y

Clear Scroll Window, set cursor to top left corner of the window.	FC58	646ØØ	-936	HOME	A,Y
Set CH=Ø, CV=(A), clear to EOP (end of page = end of window).	FC5A	646Ø2	-934		A,Y
Clear window from line (A) to blank, set cursor to left end of line (CV).	FC5C	646Ø4	-932		A,Y
Clear line (BASL), then set BASL,H from CV and WNDLFT.	FC95	64661	-875	SCRL3	A,Y
Clear line from cursor (BASL),Y, then set BASL,H from CV & WNDLFT.	FC97	64663	-873		A,Y
Load Y from CH.	& FBFØ	64496	-1Ø4Ø	STOADV	A,Y
Store A-reg to screen at (BASL),Y	& FBF2	64498	-1Ø38		A
Increment CH.	& FBF4	645ØØ	-1Ø36	ADVANCE	A
Compare (CH) with (WNDWDTH) goto CR if CH not less. Else return (RTS).	& FBF6	645Ø2	-1Ø34		A
Move cursor left one column, to right end of previous line if required and (CV) < (WNDTOP).	FC1Ø	64528	-1ØØ8	BS	A

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Move cursor up one line if (CV) < (WNDTOP).	FC1A	64538	-998	UP	A
Zero to A-reg for CH.	& FC62	6461Ø	-926	CR	A,?Y
Store A-reg to CH.	& FC64	64612	-924		A,?Y
Increment CV.	& FC66	64614	-922	LF	A,?Y
Compare CV to WNDBTM. If CV not less decrement CV and do scroll. If CV less goto VTABZ to set BASL,H and return.	FC68	64616	-92Ø		A,?Y
Place cursor at line (A) col (CH) (store A to CV and set BASL,H by JMP to VTAB).	FB5B	64347	-1189	TABV	A
Set BASL,H from CV and WNDLFT by call BASCALC and add WNDLFT.	FC22	64546	-99Ø	VTAB	A
Set BASL,H from A-reg and WNDLFT by call BASCALC and add WNDLFT.	FC24	64548	-988	VTABZ	A
Set BASL,H to memory address for left character of line in A-reg (not left character of window).	FBC1	64449	-1Ø87	BASCALC	A
Jump via CSWL, character print.	FDED	65ØØ5	-531	COUT	?A
Character print to screen output routine entry - normal for CSWL. Print character to screen with appropriate actions on controls and control characters. If (A)<$AØ goto COUTZ, bypass inverse video mask.	FDFØ	65ØØ8	-528	COUT1	?A
AUTOSTART MONITOR ONLY					
Print character to screen via VIDWAIT (pause if operator request) and VIDOUT with save and restore of A reg and Y reg.	FDF6	65Ø14	-522	COUTZ	none
OLD MONITOR ONLY					
Print character to screen via VIDOUT with save and restore of A reg and Y reg.	FDF6	65Ø14	-522	COUTZ	none

CH	36	$24	WNDLFT	32	$2Ø
CV	37	$25	WNDWDTH	33	$21
INVFLG	5Ø	$32	WNDTOP	34	$22
BASL,H	4Ø-41	$28-29	WNDBTM	35	$23

GENERAL TEXT TO THE SCREEN

The preferred method of sending text to the screen is by loading the character desired into the A-reg and calling COUT to handle it from there. The reason this is preferred is that if you want to send the output to some device other than the screen, you can change CSWL,H to point at the program supporting such other device. There are times, however, when you'll want to write to the screen regardless of the setting of CSWL,H. COUT1 is the entry point for screen-only output, where reverse video display or flashing characters are set using INVFLG. Entry at this point for the Autostart Monitor also allows you to stop output, using the control-S key.

COUTZ may be used for output to the screen without modifying the character by using INVFLG. That is, calling COUTZ with a character in the A-reg will place that character on the screen as is, without using INVFLG to display the character in inverse video or flashing mode. In the Autostart Monitor, entry at COUTZ is still early enough to handle control-S entry, stopping the system if the character being written is a Carriage Return while the keyboard buffer contains a control-S.

VIDOUT is the routine which interprets the character and places it on the screen if it is not a control character. If the VIDOUT routine is to be called directly (to bypass control-S handling in the Autostart Monitor, for example), then the calling program must save the A-reg and Y-reg before and restore the A-reg and Y-reg after, because they are both destroyed in the VIDOUT routine.

Output to the screen may be written via these alternate entry points. However, note that the Monitor will still use COUT for the keyboard input echo function, temporarily setting INVFLG to $FF for white on black for each character echoed.

Following are addresses of the above mentioned locations, and a few other entry points which will output the specified character(s) (via COUT) without the calling program having to load them into the A-reg before the call.

ADDRESS TABLE

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Print a byte to specified output device by JMP (CSWL), normally COUT1 for screen.	FDED	65ØØ5	-531	COUT	none
Character print to screen output routine entry - normal for CSWL. Print character to screen with appropriate actions on controls and control characters. If (A)<$AØ goto COUTZ, bypass inverse video mask.	FDFØ	65ØØ8	-528	COUT1	none
AND (A) with INVFLG.	& FDF4	65Ø12	-524		?A

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Print a byte to the screen. See AUTOSTART and OLD differences toward end of this table.	FDF6	65014	-522	COUTZ	none
Process char. in A-reg to screen. If control character, do control. If display character, store in screen refresh memory.	FBFD	64509	-1027	VIDOUT	A,Y
Store A-reg to screen at (BASL,H),(CH), then increment CH and goto CR if window exceeded.	FBF0	64496	-1040	STOADV	A,Y
Store A-reg to screen at (BASL,H),Y then inc CH and goto CR if window exceeded.	FBF2	64498	-1038		A
Increment CH and goto CR if window exceeded.	FBF4	64500	-1036	ADVANCE	A
Test CH. Goto CR if CH => WNDWDTH.	FBF6	64502	-1034		A
If A=$8D, $8A, $88, or $87 do it: $8D carriage return goto CR $8A line feed goto LF $88 back space goto BS $87 bell sound "bell"	FC04	64516	-1020		A
Set INVFLG to $3F = inverse video.	FE80	65152	-384	SETINV	Y
Set INVFLG to $FF = normal video.	FE84	65156	-380	SETNORM	Y
Set INVFLG from Y-reg.	FE86	65158	-378	SETIFLG	none
AUTOSTART MONITOR ONLY					
Print character to screen via VIDWAIT (stop if operator request) and VIDOUT with save and restore of A and Y regs.	FDF6	65014	-522	COUTZ	none
Test for operator pause request. If (A)=$8D (carriage return), and if keyboard register is full, and if keyboard reg contains cntl-S, then fall into KBDWAIT. Else, goto VIDOUT.	FB78	64376	-1160	VIDWAIT	Y
Pause system per operator request. Loop until new key pressed. If next key pressed is cntl-C then goto VIDOUT, leaving cntl-C in keyboard register. Else, clear keyboard strobe and goto VIDOUT.	FB88	64392	-1144	KBDWAIT	Y
OLD MONITOR ONLY					
Print character to screen via VIDOUT with save and restore of A-reg and Y-reg.	FDF6	65014	-522	COUTZ	none

CH	36	$24	WNDLFT	32	$20
CV	37	$25	WNDWDTH	33	$21
INVFLG	50	$32	WNDTOP	34	$22
BASL,H	40-41	$28-29	WNDBTM	35	$23

CONTROL CHARACTERS

Note:
The following control characters have special meanings for screen display.

$8D Carriage Return

In the Autostart Monitor, the COUTZ routine calls (JSR) VIDWAIT, which handles the control-S function before jumping to VIDOUT. The COUTZ routine in the Old Monitor calls VIDOUT.

When the VIDWAIT routine determines that the character being "written" is a Carriage Return, it then tests the keyboard input buffer for a control-S. If so, it clears the keyboard hardware for another entry and loops until another key is pressed. If this entry is other than a control-C, the keyboard strobe is cleared. Otherwise the keyboard is left filled with the control-C for the calling program to detect and handle. Then VIDWAIT JMP's to VIDOUT.

$8A Line Feed

The cursor is moved down one line unless this would put it on a line below the Scroll Window. In that case, the contents of the Scroll Window are moved up one line, and the cursor stays on the current screen line.

$88 Backspace

The VIDOUT routine moves the cursor to the left one space by decrementing CH. If CH goes negative it is set to (WNDWDTH)-1 and CV is decremented. If decrementing CV would take it above (WNDTOP) CV is not decremented. Negative scroll is not supported.

$87 Sound the Bell

The speaker is pulsed 1000 times per second for one tenth of a second.

Any other character in the range $80 thru $9F is dropped. It does not cause cursor motion or memory modification.

OUTPUT WITHOUT THE SCROLL WINDOW

If all or part of the screen is to be used in a direct addressing manner, it is necessary to avoid certain Monitor services. In general, the Scroll Window services provided by the Monitor are:

1. Scroll all text in the window up one line if a carriage return or line feed takes the cursor down through the bottom line of the window.

2. Automatically assume carriage return if window width is exceeded.

3. Place the cursor at the left edge of the Scroll Window instead of at the left edge of the screen on a carriage return.

4. Support screen clear functions:
   A. Clear the window, place cursor at top left corner.
   B. Clear the window from current cursor position.
   C. Clear line to the right of cursor position.

When using all or part of the screen as a random access display, these automatic services need be avoided.

If the full screen is to be used as a random access display, without a portion being used as a working Scroll Window, the problem is not too difficult. Consider leaving the whole screen defined as the Scroll Window.

1. The scroll operation only occurs if a carriage return or line feed or exceeding window width occurs on the bottom line of the Scroll Window. Avoid this by not having the program output CR or LF or excessive data on the bottom line of the screen, and by keeping the cursor away from the bottom line of the screen during keyboard input operations.

2. The full screen is defined as the Scroll Window by the Monitor when the RESET key is pressed. A user program can restore the window parameters to this configuration if they have been altered by calling "Set Normal Scroll Window" at $FB3C or 64316 or -122Ø.

3. Position the cursor where desired before printing a string of characters: POKE the line number into CV and call VTAB for the line and then POKE the character number into CH.

4. Output the string of characters by the same means as if operating with scroll services, being careful not to unintentionally exceed window width or output carriage returns. Depending on your screen design, however, you may intentionally do each of these.

Note that program output of a carriage return does not clear the line to the right of the carriage return, but keyboard input of a carriage return does (if reading the keyboard is being done by the Monitor get-line routines).

If part of the screen is to be allocated as an operating Scroll Window while the remainder of the screen is to be directly addressed, then a different (lower) level of Monitor services must be called upon.

One way to support a divided screen is by using the Scroll Window for data input with the Monitor get-input-line services, and by using the Scroll Window support for whatever output the program intends to put there. Then use parts of LORES graphics support for placing characters on the screen outside of the Scroll Window, as described below. The aim here is to leave support of cursor position (zero page fields CV, CH, and BASL,H) up to the Monitor, and use other methods/fields for placing characters outside the Scroll Window.

To place characters outside the Scroll Window,

1. With the line number in the A-reg, call GBASCALC to set GBASL,H to point to the memory address of the left character position of the indicated screen line.

2. With Y-reg indicating horizontal position on the line, store the desired character at (GBASL),Y.

Note that this technique does not interfere with LORES plotting if the screen is being used in mixed mode, because PLOT calls always set GBASL,H as required without regard to possible previous contents.

Another approach is available for the BASIC or APPLESOFT programmer. Again, the Scroll Window support can be used for some things, while the following approach can be used to place characters on the screen outside of the window. That approach is to compute the screen memory location for each byte to the screen, and poke the byte there. A variation on that approach is shown by the sample program. In the sample, the Monitor VTAB routine is used to assist in building a table of memory locations indicating the starting points of the screen lines. This is an easier alternative than using the modulo arithmetic formula described in the section "Pages Four thru Eleven". Note that adding 1Ø24 to each value in the table gives the memory address for that line in the secondary display area.

ADDRESS TABLE

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
OUTSIDE OF SCROLL WINDOW					
Compute memory address for line in A-reg; set GBASL,H.	F847	63559	-1977	GBASCALC	A
INSIDE SCROLL WINDOW					
Write byte in A-reg to screen at cursor (CV),(CH) using INVFLG and supporting cursor move.	FDFØ	65ØØ8	-528	COUT1	?A
Write byte in A-reg to screen at (CV),(CH) with cursor move but not INVFLG.	FDF6	65Ø14	-522	COUTZ	none
Clear Scroll Window to blanks, cursor to top left corner.	FC58	646ØØ	-936	HOME	A,Y
Set CV from A-reg, clear window to end of window.	FC5A	646Ø2	-934		A,Y

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Place cursor at line (A) col (CH) setting CV and BASL,H from A-reg.	FB5B	64347	-1189	TABV	A
Set BASL,H from CV (and WNDLFT).	FC22	64546	-990	VTAB	A
Set BASL,H from (A) and WNDLFT without regard to CV.	FC24	64548	-988	VTABZ	A
Set BASL,H to left end of screen line (not window line) in A-reg.	FBC1	64449	-1087	BASCALC	A

CH	36	$24	WNDLFT	32	$20
CV	37	$25	WNDWDTH	33	$21
GBASL,H	38-39	$26-27	WNDTOP	34	$22
BASL,H	40-41	$28-29	WNDBTM	35	$23
INVFLG	50	$32			

APPLESOFT SAMPLE PROGRAM

```
10      REM  TEXT OUTPUT WITHOUT THE SCROLL WINDOW
11      REM  SAMPLE PROGRAM
12      REM  READS FROM KEYBOARD  LINE, CHAR, STRING
14      REM  AND PLACES THE STRING THERE
1000    REM  PROGRAM ENTRY
1010    DIM L%(23):        REM  LINE ADDR TABLE
1100    GOSUB 63000:       REM  MAKE UP TABLE
1199                       REM  PRINT PART OF TABLE JUST FOR SHOW
1200    FOR I = 0 TO 21: PRINT I,L%(I): NEXT
1209                       REM  DELAY TO ALLOW LOOK AT IT.
1210    FOR I = 1 TO 5000: NEXT
1220    PRINT:      REM  PRINT CR TO ALLOW CTL-S STOP IF DESIRED.
1225    CALL  - 936:       REM CLEAR SCREEN BEFORE CHANGING WINDOW.
1229                       REM  SET UP NEW WINDOW.
1230    POKE 32,24: POKE 33,14: POKE 34,12: POKE 35,17
1235    CALL -936:         REM  PUT CURSOR INTO WINDOW AREA.
1300    INPUT LI,CL,SS$:REM  READ A COMMAND LINE.
1399                       REM  ALLOW A WAY OUT
1400    IF SS$ = "END" THEN 63900
1500    SL =  LEN (SS$)
1509                       REM CHECK LEGALITY OF LINE, ETC.
1510    IF LI > 23 THEN 1810
1511    IF CL > 39 THEN 1810
1519                       REM NOT PAST 40 THOUGH.
1520    IF CL + SL > 39 THEN SL = 40 - CL
1600                       REM  PUT CHARACTERS ONE AT A TIME.
1601    FOR I = 1 TO SL
1700    C$ =  MID$ (SS$,I,1):C% =  ASC (C$)
1720    POKE L%(LI) + CL + I - 1,C% + 128
1740    NEXT I
1800    GOTO 1300:         REM GO BACK FOR ANOTHER COMMAND.
```



```
1810                       REM  LINE OR CH TOO BIG - ERROR.
1811    CALL  - 936: PRINT "NOT SO BIG"
1812    PRINT "LN ";LI: PRINT "CH ";CL
1820    GOTO 1800
62999                      REM
63000                      REM  MAKE UP LINE ADDRESS TABLE
63010   X% =  PEEK (37): REM  REMEMBER CV
63020   FOR I = 0 TO 23
63030   POKE 37,I:         REM  SET CV
63031   CALL -990:         REM  CALL VTAB TO FILL BASL & BASH
63035   L%(I) = 256 * ( PEEK (41)) +  PEEK (40)
63040   NEXT I
63045                      REM  TABLE SETUP DONE
63046                      REM  RESTORE CV AND RETURN
63050   POKE 37,X%: CALL  - 990: REM  WITH PROPER BASL & BASH
63060   RETURN
63900   CALL  - 1233: END: REM RESTORE FULL WINDOW PRIMARY
```

SECONDARY DISPLAY AREAS

The Apple II hardware allows use of either of two memory areas for display to the screen. The first, or primary, is memory locations $0400-$07FF. The secondary text (and low resolution graphics) display area is $0800-$0BFF. This area is normally overlaid by a user program or data, but in special circumstances a user may desire to make use of this secondary area as a screen display area.

The Monitor does not support the secondary display area as such. That is, the routines in the Monitor which determine screen area memory address from line number (CV) and character column (CH) do so only for the primary display area. These routines perform correctly only for lines 0-23.

Following are descriptions of two ways of using the secondary display area.

COPY PRIMARY TO SECONDARY

There are times when it is desirable to change the display very quickly, although the program produces the output slowly. For example, a program might display data found by scanning a disk file. The programmer might generate the original screen data in the primary display area, then move it to the secondary display area and set the hardware to display from secondary. The program may then proceed to generate the next screen data in the primary area while the operator is looking at the initial or previous display of results. A sample program is provided later in this section showing how the Monitor Move routine can be used to move the contents of the primary display area to the secondary display area.

SET BASL,H FOR SECONDARY DISPLAY PAGE

When the Monitor places a character in the screen memory area, it does so using BASL,H as the address of the memory location for the left end of the line, and (CH) as the displacement from the left end of the line. BASL,H can be initialized to the memory location of a selected screen line by setting the desired line number in CV and then CALLing TABV. On return from that CALL, adding 4 to BASH changes BASL,H to point to memory for the desired line in the secondary display area. This will last until the program writes a carriage return or writes characters beyond the right end of the Scroll Window.

If the Monitor is called upon to read from the keyboard, it "echoes" the input characters to the screen. Input of a carriage return, one backspace too many, a cursor movement, or a screen clearing Escape Key function will cause BASL,H to be restored by the Monitor to point within the primary display area.

In the case where one display area is to be used for text and the other for graphics, it is preferable to keep the graphics in the primary area and the text in the secondary area because the Monitor recomputes GBASL,H continually for plotting functions, whereas for text output BASL,H is recomputed only when it is necessary to move the cursor to a new line.

It must be noted that APPLESOFT also does not (easily) support the secondary display area. APPLESOFT in RAM occupies that part of memory, and then some. Firmware APPLESOFT places the program code in that memory space, unless special actions are taken. Those actions may be noted in the sample program, which uses APPLESOFT and the secondary display area. POKE 1Ø4,12 and 3Ø27,Ø before loading the program.

ADDRESS TABLE

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Place cursor at line (A), col (CH) (store A to CV and compute BASL,H by JMP to VTAB.	FB5B	64347	-1189	TABV	A
Set BASL,H from CV and WNDLFT by call BASCALC and add WNDLFT.	FC22	64546	-99Ø	VTAB	A
Set BASL,H from A-reg and WNDLFT by call BASCALC and add WNDLFT.	FC24	64548	-988	VTABZ	A
Set BASL,H to memory address for left character of screen (not window) of line in A-reg.	FBC1	64449	-1Ø87	BASCALC	A

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Write byte in A-reg to screen at cursor (CV),(CH) using INVFLG and supporting cursor move.	FDFØ	65ØØ8	-528	COUT1	?A
Write byte in A-reg to screen at (CV),(CH) with cursor move but not INVFLG.	FDF6	65Ø14	-522	COUTZ	none
Monitor Command Processor MOVE routine. (A1L,H) thru (A2L,H) is moved to (A4L,H) thru whatever.	FE2C	65Ø68	-468	MOVE	A (Y=Ø)
Monitor Command Processor GO entry. Set PCL,H from A1L,H if entered. &	FEB6	652Ø6	-33Ø	GO	A,X,Y,P
Call RESTORE, set all regs but S &	FEB9	652Ø9	-327		
JMP via PCL,H.	FEBC	65212	-324		

DIRECT CONTROL ADDRESSES

The following table describes the methods of setting the hardware for display to various screen configurations by direct control rather than by calling the Monitor. For some of these items there is no routine in the Monitor which could be called to perform the function.

Function	Method
Set GRAPHICS display mode.	POKE -163Ø4,Ø or STA CØ5Ø
Set TEXT display mode.	POKE -163Ø3,Ø or STA CØ51
Set Graphics mode to Full Screen.	POKE -163Ø2,Ø or STA CØ52
Set MIXED GRAPHICS and TEXT mode.	POKE -163Ø1,Ø or STA CØ53
Set display to Primary Page.	POKE -163ØØ,Ø or STA CØ54
Set display to Secondary Page.	POKE -16299,Ø or STA CØ55
Clear HIRES = Set LORES mode.	POKE -16298,Ø or STA CØ56
Set HIRES Graphics mode.	POKE -16297,Ø or STA CØ57
Set top line of Scroll Window.	POKE 34,line-number (Ø-23) Bottom must be greater than top.
Set left edge of Scroll Window.	POKE 32,column-number (Ø-39) Left edge + width not to exceed 4Ø.
Set width of Scroll Window.	POKE 33,number-of-columns (1-4Ø), Left edge + width not to exceed 4Ø.
Set bottom line of Scroll Window.	POKE 35,line-number (1-24) Bottom must be greater than top.

CH	36	$24	WNDLFT	32	$2Ø
CV	37	$25	WNDWDTH	33	$21
GBASL,H	38-39	$26-27	WNDTOP	34	$22
BASL,H	4Ø-41	$28-29	WNDBTM	35	$23
INVFLG	5Ø	$32			

INTEGER BASIC SAMPLE PROGRAM

```
10 REM  SAMPLE SECONDARY DISPLAY WAY
11 REM  USING MONITOR MOVE TECHNIQUE

19 GOTO 1000:                  REM  BYPASS SUBROUTINES

20                  REM  MOVE AREA 1 TO AREA 2
21 POKE 60,0: POKE 61,4:       REM SET A1L,H
22 POKE 62,255: POKE 63,7:     REM SET A2L,H
23 POKE 66,0: POKE 67,8:       REM SET A4L,H
25 POKE 71,0:                  REM SET Y-REG=0
26 POKE 58,44:                 REM $2C
27 POKE 59,254:                REM $FE
28 CALL -327:                  REM DO THE MOVE
29 RETURN

1000                 REM  PROGRAM START
1001 IF PEEK (75)<12 THEN 32000

1100 CALL -936:                REM  CLEAR THE SCREEN

1200 PRINT "THIS IS THE SECONDARY DISPLAY AREA"
1210 PRINT "NOTE THE LACK OF CURSOR"

1300 GOSUB 20:                 REM  MOVE TO SECONDARY

1400 CALL -936:                REM  CLEAR PRIMARY AGAIN
1410 PRINT "THIS IS THE PRIMARY AREA AGAIN"
1500 POKE -16299,0:            REM  SET SECONDARY
1600 FOR I=I TO 4000: NEXT I
1700 POKE -16300,0:            REM  BACK TO PRIMARY
1800 END

32000              REM  NO LOMEM ERROR
32001 PRINT "PLEASE LOAD AGAIN"
32002 PRINT "AFTER LOMEM:3072 "
32003 END
```

APPLESOFT SAMPLE PROGRAM

```
10           REM SECONDARY DISPLAY AREA WAYS AND MEANS
11           REM SAMPLE PROGRAM
12           REM READS FROM KEYBOARD
13           REM COMMAND, LINE, CHARACTER, STRING
14           REM AND PLACES THE STRING

1000           REM PROGRAM ENTRY

1009           REM IS SECONDARY AREA CLEAR?
1010   IF  PEEK (104) < 12 THEN 62000

1020   GOSUB 63000:               REM CLEAR THE SECONDARY
```



```
                    REM  M A I N   P R O G R A M
1300   IF Q = 0 THEN 1390:        REM INPUT TO PRIMARY
1309                              REM SET INPUT TO SECONDARY
1310   POKE 37,21:                REM SET LINE 21
1311   CALL  - 990:               REM SET BASL,H
1312   POKE 41, PEEK (41) + 4:    REM SET BASH TO SECONDARY

1390   INPUT CC$,LI,CL,SS$
1400   IF CC$ = "END" THEN 63900
1410   IF CC$ = "S" THEN 2000:  REM SET SHOW TO SECONDARY AREA
1420   IF CC$ = "P" THEN 2100:  REM SET SHOW TO PRIMARY AREA
1430   IF CC$ = "Q" THEN 2200:  REM SET INPUT SECONDARY
1431   IF CC$ = "R" THEN 2300:  REM SET INPUT PRIMARY
1440   IF CC$ = "X" THEN 1500:  REM PUT STRING TO SECONDARY
1450   POKE 16300,0: PRINT "WHAT? ": GOTO 1300
1500   SL =  LEN (SS$)
1510   IF LI > 23 THEN 1810
1511   IF CL > 39 THEN 1810
1512   IF CL + SL > 39 THEN SL = 40 - CL: REM NO AUTO CR
1590   CX =  PEEK (37):           REM REMEMBER CV
1600   POKE 37,LI: CALL  - 990: POKE 41, PEEK (41) + 4
1610   POKE 37,CX:                REM RESTORE CV
1620   POKE 36,CL:                REM SET CH FOR THIS PRINT
1700   SP$ =  LEFT$ (SS$,SL):     REM SHORTEN PRINT IN THIS SMPL
1710   PRINT SP$
1800   GOTO 1300
1810   CALL  - 936:               REM VALUE TOO LARGE.
1811   PRINT "NOT SO BIG":        REM PRINT IN PRIMARY ONLY
1812   PRINT "LN ";LI: PRINT "CH ";CL
1820   GOTO 1300

2000   POKE  - 16299,0:           REM SET SECONDARY
2010   GOTO 1300

2100   POKE  - 16300,0:           REM SET PRIMARY
2110   GOTO 1300

2200   Q = 1: GOTO 1300:          REM SET INPUT TO SECONDARY
2300   Q = 0: GOTO 1300:          REM SET INPUT TO PRIMARY
```

```
62ØØØ  PRINT "SETUP NOT MADE, NOW BEING DONE"
62Ø1Ø  PRINT "RUN THE PROGRAM AGAIN"
62Ø18            REM 1Ø4 IS APPLESOFT ROM START
62Ø19         REM BYTE BEFORE $CØ1 MUST BE ZERO
62Ø2Ø  POKE 3Ø72,Ø: POKE 1Ø4,12: END

63ØØØ  BL$ = "     ":              REM CLEAR SECONDARY AREA
63ØØ1  FOR I = 1 TO 3:BL$ = BL$ + BL$: NEXT
63ØØ5  CX =  PEEK (37)
63Ø1Ø  FOR I = Ø TO 23
63Ø2Ø  POKE 37,I: CALL  - 99Ø
63Ø3Ø  POKE 41, PEEK (41) + 4
63Ø4Ø  POKE 36,Ø
63Ø5Ø  PRINT BL$
63Ø6Ø  NEXT
63Ø7Ø  POKE 37,CX: POKE 36,Ø
63Ø8Ø  RETURN
639ØØ  POKE 163ØØ,Ø: CALL  - 1233: END
```


CHAPTER 3

INTERRUPT PROCESSING

Some computers are capable of reacting to the raising (or dropping) of a signal line by instantly saving the current status of the processor, and quickly transferring control to some other program within the computer. Changing the state of that line is called "causing an interrupt". The functions of the processor in saving its current state and transferring control to some other location in memory is called "taking an interrupt". The program which then receives control is expected to "handle the interrupt".

The 65Ø2 microprocessor in the Apple II is sensitive to three interrupt categories. These are RESET, NMI (Non-Maskable Interrupt), and IRQ. Execution of a BRK instruction causes a form of IRQ interrupt to be simulated.

The purpose of an interrupt, in general, is to allow some kind of external device to make a condition known to a running program without the program having to periodically or continually test for the hardware condition. An example of the latter type of operation is the Apple II keyboard operation. When keyboard input is to be accepted memory location $CØØØ is tested repeatedly until presence of the sign bit indicates that a key has been pressed. An example of interrupt driven processing could be a special peripheral controller card, attached to a telephone line, which caused the computer to be taken over by a data acquisition program any time data was available, but would allow the machine to be used for other things in between transmissions.

When a computer recognizes (takes) an interrupt, the hardware should accomplish three things.

1. Save processor status in such a way that execution of the interrupted program can be continued after the interrupt has been "serviced" or handled.

2. Prevent further recognition of that class of interrupts until the interrupt handling program restores that interruptability.

3. Transfer control to the program meant to handle this type or category of interrupt.

With the 65Ø2 in the Apple II variations on the above three steps are taken for the three different interrupt classes or categories.

1. When an IRQ (or BRK) or NMI interrupt is taken, the contents of the program counter and the P-reg (processor status register) are respectively pushed onto the stack. When a RESET interrupt is taken, the processor holds the memory in READ mode until control is transferred to the handler, so nothing of processor status is pushed onto the stack.

2. When the 65Ø2 takes an IRQ interrupt, the P-reg is modified. If a BRK instruction is executed, the $1Ø bit of the processor status register is set to one before the P-reg is pushed onto the stack. If the IRQ line was the cause of the interrupt, this bit is set to zero before the P-reg is pushed onto the stack.

   After the P-reg is pushed onto the stack, the $Ø4 bit is set to inhibit recognition of any more IRQ category interrupts until the interrupt handling program clears this condition.

   With RESET and NMI there is no available facility for preventing another interrupt while the current interrupt is being handled.

3. The 65Ø2 transfers control to the appropriate program for handling an interrupt by means of "vectors". Memory addresses $FFFA-$FFFF are reserved for this purpose. The final step of taking an interrupt is loading of the program counter from the vector for this class or category of interrupt. The following table indicates the locations of the interrupt handlers for the two Monitors.

Interrupt Taken	Vector Address	Monitor Label	Old Monitor Address	Autostart Address
NMI	$FFFA-B	"NMI"	$Ø3FB	$Ø3FB
RESET	$FFFC-D	RESET	$FF59	$FA62
IRQ/BRK	$FFFE-F	IRQ	$FA86	$FA4Ø

NMI INTERRUPT

The Apple II Monitor does not interfere with user handling of the NMI interrupt. That is, the vector for NMI causes the 65Ø2 to transfer control of the computer to location $Ø3FB, where the user is to place a JMP to the user-provided handler for this type of interrupt.

RESET INTERRUPT SUPPORT

Pressing the RESET key on the keyboard causes a RESET interrupt to occur. On all Apple II's but the very early ones, power-on also results in generation of a RESET interrupt.

The actions performed by the Autostart Monitor and the Old Monitor RESET interrupt handlers are considerably different. Therefore, they will be described separately.

IRQ/BRK INTERRUPT HANDLING

When either an IRQ interrupt is taken or a BRK instruction is executed, the 65Ø2 performs an interrupt sequence. The contents of the program counter are pushed onto the stack. The $1Ø bit of the P-reg is set or cleared to indicate the IRQ line vs. BRK instruction, and then it is pushed onto the stack. The 65Ø2 then sets the $Ø4 bit of the P-reg, preventing another interrupt of this type from being recognized until this one is handled. The 65Ø2 then loads the Program Counter from the IRQ hardware prescribed vector at $FFFE-$FFFF, and allows operation of the computer to continue from that point. The interrupt Handler for IRQ interrupts is now in control.

RESET INTERRUPT—OLD MONITOR

When a RESET interrupt is taken the Old Monitor establishes a predefined configuration of hardware and page zero fields. Primarily, the keyboard is set as the current input device, the screen is set as the current output device, and the screen configuration is set to full screen Scroll Window with normal video.

Page zero fields KSWL,H, CSWL,H are set to make the keyboard and screen active. WNDLFT, WNDWDTH, WNDTOP, WNDBTM are set to define the whole screen as the Scroll Window. CV and CH are set to place the cursor at the bottom left corner of the screen. INVFLG is set to normal (white on black).

Hardware addresses are referenced to establish a known configuration as follows.

$CØ56 - clear high resolution graphics
$CØ54 - display primary area
$CØ51 - set text mode

Control is then transferred to the "top" of the Monitor at label MON, location $FF65, at which point the "bell" is sounded and the Monitor enters the command line read routine.

ADDRESS TABLE

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Set STATUS in SAVE area to Ø.	& FB2F	643Ø3	-1233	INIT	A
Clear HIRES.	& FB33	643Ø7	-1229		A
Set primary display area.	& FB36	6431Ø	-1226		A
Set TEXT mode.	& FB39	64313	-1223	SETTXT	A
Set full screen scroll window by branch to SETWND with (A)=Ø.	FB3C	64316	-122Ø		A
Set WNDTOP from A-reg.	& FB4B	64331	-12Ø5	SETWND	A
Load A with Ø for WNDLFT.	& FB4D	64333	-12Ø3		A
Set WNDLFT from A-reg.	& FB4F	64335	-12Ø1		A
Load A with 4Ø for WNDWDTH.	& FB51	64337	-1199		A
Set WNDWDTH from A-reg.	& FB53	64339	-1197		A
Load A with 24 for WNDBTM.	& FB55	64341	-1195		A
Set WNDBTM from A-reg.	& FB57	64343	-1193		A
Load A with 23 for CV.	& FB59	64345	-1191		A
Set CV from A-reg.	& FB5B	64347	-1189	TABV	A
JMP to VTAB to set BASL,H & RTS.	FB5D	64349	-1187		A
Set INVFLG to $FF = normal video.	FE84	65156	-38Ø	SETNORM	Y
Set INVFLG from Y-reg.	FE86	65158	-378	SETIFLG	none
Set port Ø (keyboard) for input.	FE89	65161	-375	SETKBD	A,X,Y
Set port Ø (screen) for output.	FE93	65171	-365	SETVID	A,X,Y
Monitor entry on RESET key pressed or Power on.	FF59	65369	-167	RESET	
Call SETNORM - white on black.	&				
Call INIT -Text & full scroll.	& FF5C	65372	-164		
Call SETVID - screen as output.	& FF5F	65375	-161		
Call SETKBD - keyboard = input.	& FF62	65378	-158		
Clear 65Ø2 decimal mode (set hex)	& FF65	65381	-155	MON	
Sound bell.	& FF66	65382	-154		
Monitor Command Processor Entry. Set "*" as prompt character.	FF69	65385	-151	MONZ	

RESET INTERRUPT—AUTOSTART MONITOR

The Autostart Monitor performs functions of three categories in handling a RESET interrupt.

1. Establish a known hardware/software environment with regards to the basic machine.

2. If the contents of memory (page three) do not indicate that a power-on initialization has been performed, the Autostart Monitor will perform power-on initialization. If a disk controller card is present in one of the slots, power-on initialization includes bootstrapping from that slot. If no disk controller card is in the machine a control-B entry is simulated. In either case, the appropriate language processor receives control at the end of power-on initialization, with page three fields set to indicate that a warm start is to be performed on ensuing interrupts from the RESET key.

3. If the contents of memory (page three) indicate that power-on initialization has already been performed, the Autostart Monitor will transfer control via the RESET (Soft Entry) vector in page three at the conclusion of "handling" the RESET interrupt. If DOS has been booted, this will result in transfer of control back to the current language processor through DOS. If DOS is not present, the normal setting of the RESET vector will cause simulation of a control-C (warm start) reentry into the current language.

INITIALIZE SYSTEM CONFIGURATION

When a RESET Interrupt is taken, the Autostart Monitor establishes a predefined configuration of hardware and page zero fields. Primarily, the keyboard is set as the current input device, the screen is set as the current output device, and the screen configuration is set to full screen Scroll Window with normal video.

Page zero fields KSWL,H, CSWL,H are set to make the keyboard and screen active. WNDLFT, WNDWDTH, WNDTOP, WNDBTM are set to define the whole screen as the Scroll Window. CV and CH are set to place the cursor at the bottom left corner of the screen. INVFLG is set to normal (white on black).

Hardware addresses are referenced to establish a known configuration as follow.

$CØ56 - clear high resolution graphics
$CØ54 - display primary area
$CØ51 - set text mode
$CØ58 - clear ANØ = TTL LO
$CØ5A - clear AN1 = TTL LO
$CØ5D - set AN2 = TTL HI
$CØ5F - set AN3 = TTL HI
$CFFF - turn off Expansion ROM
$CØ1Ø - clear keyboard strobe

On completion of all the above, the Autostart Monitor sounds the BELL.

COLD/WARM DETERMINATION

After establishing a known basic hardware and software (screen control) environment, the Autostart Monitor executes a test to determine whether power-on initialization is to be performed. Page three locations $Ø3F2-$Ø3F3 contain the RESET (Soft Entry) vector, the address to which the Autostart Monitor will transfer control on completion of handling the RESET interrupt. Location $Ø3F4 is a

validation byte, used with $Ø3F3 to indicate whether or not power-on initialization is to be performed. If the Exclusive OR of the contents of these two memory locations is $A5, then power-on initialization is considered to have been previously accomplished, and $Ø3F2-$Ø3F3 is considered a valid address to which to transfer control.

POWER-ON INITIALIZATION

The first functions of power-on initialization are to establish in page three ($Ø3FØ-$Ø3F4) the BRK interrupt vector (see "BRK Instruction Handling - Autostart Monitor") and the RESET Soft Entry interrupt vector with validation byte. The RESET vector at this point is set to $EØØØ to simulate a control-B (initialize) entry for the current language processor.

The Autostart Monitor next performs a routine which tests each slot, from slot 7 through slot 1, for presence of a disk controller card. If one is found, a jump is performed to $CXØØ where X is the slot number in which the disk controller has been found. This will result in loading of DOS and presumably execution of the HELLO program.
Note: DOS 3.2 Replaces the RESET vector at $Ø3F2-$Ø3F3 and validation byte at $Ø3F4, so that on a RESET interrupt, control will be passed through DOS back to the current language processor.

If no disk controller card is found the Autostart Monitor changes the RESET vector to $EØØ3 (language restart or control-C entry point) and then jumps to $EØØØ (language initialize entry point).

SYSTEM RESTART

If the $Ø3F3-$Ø3F4 test described above is passed, the RESET vector at $Ø3F2-$Ø3F3 is considered mostly valid. If it contains $EØØØ, it is changed to $EØØ3 and then BASIC is entered at $EØØØ. If it is not $EØØØ, it executes an Indirect Jump via $Ø3F2-$Ø3F3 to the address specified therein.

RESET VECTOR MODIFICATION BY USER

The RESET vector may be modified by user or program to send control to some other address in the machine at the completion of Monitor handling of the interrupt. For example, to cause the RESET key to result in placing the machine in Monitor mode, execute the following program;

```
1Ø POKE 1Ø1Ø,1Ø5
2Ø POKE 1Ø11,255
3Ø POKE 1Ø12,9Ø
4Ø CALL -151: REM ENTER MONITOR
5Ø END
```


The following program is more general purpose. In order to set the RESET vector to some address, poke the address into locations 1Ø1Ø-1Ø11 ($Ø3F2-$Ø3F3) and then CALL Autostart Monitor label SETPWRC ($FB6F or 64367 or -1169) to set location 1Ø12 ($Ø3F4).

```
1Ø REM AD IS ADDRESS OF
11 REM ROUTINE TO RECEIVE
12 REM CONTROL AFTER RESET
2Ø POKE 1Ø1Ø,AD: REM SET LO BYTE
3Ø POKE 1Ø11,AD/256: REM SET HI
4Ø CALL -1169: REM SET 1Ø12
```

Note: If you try to run this on a system with an Old Monitor ROM, you may destroy the program, or even the entire diskette. To avoid this problem, execute the steps in the above program manually, on a system with an Autostart ROM. Then, PEEK location 1Ø12 and get the value to POKE into 1Ø12, alleviating the need to CALL-1169 at all.

ADDRESS TABLE

Function		Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Monitor entry on RESET key pressed or Power on.		FA62	64Ø98	-1438	RESET	
CLD - clear 65Ø2 dec,(set hex).	&					
Call SETNORM - white on black.	&	FA63	64Ø99	-1437		
Call INIT - Text, full scroll.	&	FA66	641Ø2	-1434		
Call SETVID - screen as output.	&	FA69	641Ø5	-1431		
Call SETKBD - keyboard as input	&	FA6C	641Ø8	-1428		
Initialize hardware to known state.		FA6F	64111	-1425	INITAN	
Clear ANØ to TTL LO (ref. CØ58).	&					
Clear AN1 to TTL LO (ref. CØ5A).	&	FA72	64114	-1422		
Set AN2 to TTL HI (ref. CØ5D).	&	FA75	64117	-1419		
Set AN3 to TTL HI (ref. CØ5F).	&	FA78	6412Ø	-1416		
Clear Expansion ROM (ref. CFFF).	&	FA7B	64123	-1413		
Clear keyboard strobe.	&	FA7E	64126	-141Ø		
Clear 65Ø2 decimal mode (set hex).	&	FA81	64129	-14Ø7	NEWMON	
Call BELL.	&	FA82	6413Ø	-14Ø6		
Test $3F3 vs. $3F4: Cold or Warm. If Cold goto PWRUP. If ($3F3) XOR ($3F4) = $A5, Warm.		FA85	64133	-14Ø3		
Test SOFTEV ($3F2) low byte: Non-zero means Cold Start done - Goto NOFIX to use SOFTEV vector. Zero means restart warm maybe.		FA8F	64143	-1393		
Test SOFTEV hi for $EØ - language cold start entry. If not equal, SOFTEV is ok to use, goto NOFIX.		FA94	64148	-1388		
SOFTEV = $EØØØ, change to $EØØ3 for future use and goto $EØØØ to cold start the language.		FA9B	64155	-1381	FIXSEV	

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
JMP (SOFTEV): Use the Soft Entry vector to exit RESET handler.	FAA3	64163	-1373	NOFIX	
Cold Start on RESET entry point. Call APPLEII to clear screen and put title on top line.	& FAA6	64166	-137Ø	PWRUP	
Set page 3 interrupt vectors for BRK (OLDBRK) and SOFTEV ($EØØØ).	FAA9	64169	-1367	SETPC3	
Look for disk controller card in slots 7 thru 1. If none, goto FIXSEV above to set SOFTEV for BASIC restart & enter BASIC cold. If disk found, JMP (LOCØ) to boot from the disk.	FAB4	6418Ø	-1356		
Clear screen (call HOME).	& FB6Ø	64352	-1184	APPLEII	A,Y
Place APPLE II legend on top line.	FB63	64355	-1181		A,Y
Set PWREDUP ($3F4) = ($3F3) XOR $A5	FB6F	64367	-1169	SETPWRC	A
Set STATUS in SAVE area to Ø.	& FB2F	643Ø3	-1233	INIT	A
Clear HIRES.	& FB33	643Ø7	-1229		A
Set primary display area.	& FB36	6431Ø	-1226		A
Set TEXT mode.	& FB39	64313	-1223	SETTXT	A
Set full screen scroll window by branch to SETWND with (A)=Ø.	FB3C	64316	-122Ø		A
Set WNDTOP from A-reg.	& FB4B	64331	-12Ø5	SETWND	A
Load A with Ø for WNDLFT.	& FB4D	64333	-12Ø3		A
Set WNDLFT from A-reg.	& FB4F	64335	-12Ø1		A
Load A with 4Ø for WNDWDTH.	& FB51	64337	-1199		A
Set WNDWDTH from A-reg.	& FB53	64339	-1197		A
Load A with 24 for WNDBTM.	& FB55	64341	-1195		A
Set WNDBTM from A-reg.	& FB57	64343	-1193		A
Load A with 23 for CV.	& FB59	64345	-1191		A
Set CV from A-reg.	& FB5B	64347	-1189	TABV	A
Jump to VTAB to set BASL,H & RTS.	FB5D	64349	-1187		A
Set INVFLG to $FF = normal video.	FE84	65156	-38Ø	SETNORM	Y
Set INVFLG from Y-reg.	FE86	65158	-378	SETIFLG	none
Set port Ø (keyboard) for input.	FE89	65161	-375	SETKBD	A,X,Y
Set port Ø (screen) for output.	FE93	65171	-365	SETVID	A,X,Y
FOR COMPATIBILITY WITH OLD MONITOR the RESET routine is still here. Call SETNORM - white on black.	FF59 &	65369	-167	OLDRST	
Call INIT -Text & full scroll.	& FF5C	65372	-164		
Call SETVID - screen as output.	& FF5F	65375	-161		
Call SETKBD - keyboard = input.	& FF62	65378	-158		
Clear 65Ø2 decimal mode,set hex.	& FF65	65381	-155	MON	
Sound bell.	& FF66	65382	-154		
Monitor Command Processor Entry. Set "*" as prompt character.	FF69	65385	-151	MONZ	

IRQ/BRK INTERRUPTS

IRQ/BRK INTERRUPT RECOGNITION

When either an IRQ interrupt is taken or a BRK instruction is executed the 65Ø2 performs an interrupt sequence. The contents of the program counter are pushed onto the stack. The $1Ø bit of the P-reg is set or cleared in indication of IRQ line vs. BRK insrruction, and then it is pushed onto the stack. The 65Ø2 then sets the $Ø4 hit of P-reg, preventing another interrupt of this type from being recognized until this one is handled. The 65Ø2 then loads the Program Counter from the IRQ hardware prescribed vector at $FFFE-$FFFF, and allows operation of the computer to continue from that point. The Interrupt Handler for IRQ interrupts is now in control.

IRQ INTERRUPT HANDLING

The 65Ø2 directing vector at $FFFE-$FFFF points to Monitor program label IRQ in both the Old Monitor and the Autostart Monitor. It will be noted in the address table that the address is different, however.

The handling of an IRQ interrupt is identical in hoth Monitors. The contents of the A-reg are stored at ACC ($45) for future reference. The processor status (P-reg) pushed onto the stack by the taking of the interrupt is popped into the A-reg, and then pushed back onto the stack so that the stack and pointer are not changed. By shifting the A-reg left three bits, the IRQ routine moves into the sign bit the bit which indicates (in this case by being a zero) that the interrupt is an IRQ interrupt rather than execution of a BRK instruction. The Monitor then executes a Jump Indirect instruction via location $Ø3FE-$Ø3FF to the user provided IRQ Interrupt Handler. Note that on an IRQ interrupt the X, Y, and S registers are not saved by the Monitor. Also, the interrupt handler has the responsibility of clearing the $Ø4 bit on exit to allow further interrupts.

BRK INSTRUCTION INTERRUPT

Execution of a BRK instruction causes the 65Ø2 to simulate an IRQ interrupt with minor changes. Due to the method the instruction is handled, the address pushed onto the stack as part of the Interrupt simulation is two bytes beyond the BRK instruction executed.

Before pushing the P-reg onto the stack, the $1Ø bit is ser to indicate to the interrupt handling routine that the cause of the interrupt was execution of a BRK instruction rather than the IRQ line. After pushing the P-reg onto the stack, the $Ø4 bit is set to inhibit IRQ interrupts from being recognized until the interrupt handler clears the condition. Control is then transferred according to the 65Ø2 IRQ interrupt vector to Monitor label IRQ. As described above regarding handling of an IRQ interrupt, the IRQ routine first stores the A-reg at ACC ($45) for future reference, and then uses the A-reg to test the stacked P-reg contents for a one in the $1Ø position. The stack and stack pointer are not changed by this operation. The result

of the test is a transfer of control to Monitor label BREAK. Note in the address table that the address of BREAK is not the same in the two Monitors.

BRK INSTRUCTION—SAVING OF STATUS

In each Monitor the first thing done in the BREAK routine is to save full machine status in page zero. The contents of the A-reg have already been stored by entry into the IRQ interrupt handler. The BREAK routine pops the stacked contents of the P-reg from the stack, and does a JSR to SAV1 at which point the remaining registers are saved. Note that this clears the $04 bit, allowing further IRQ or BRK interrupts to be taken. The S-reg saved at that time, however, has been incremented once by popping the P-reg back from the stack and decremented twice by the JSR to SAV1. On return from SAV1, the BREAK routine pops the Program Counter from the stack and stores it in page zero locations PCL-PCH. The address table at the end of this section indicates the page zero locations at which the above items are stored.

BRK INSTRUCTION—OLD MONITOR

The function of the BRK instruction interrupt handler of the Old Monitor is to display through COUT the machine status at the time the BRK instruction was encountered, and then return control to the top of the Monitor at label MON. The details above describe the handling of the interrupt through storage of machine status in page zero, including PCL,H. The Old Monitor BREAK routine next does a JSR to INSDS1 to display the instruction at the address indicated by PCL-PCH (which is two bytes beyond the BRK executed), and a JSR to RGDSP1 to display the contents of the five registers, P, A, X, Y, S. Note that the S-reg as displayed is two less than it was at the time of the BRK execution due to the JSR to SAV1. On completion of the register display, a JMP to MON completes the handling of the interrupt.

BRK INSTRUCTION—AUTOSTART MONITOR

The Autostart Monitor handles IRQ interrupt which is really a BRK instruction interrupt by saving registers and Program Counter in page zero locations. The Autostart Monitor BREAK routine then exits via the Apple-II BREAK vector at $03F0-$03F1. Thus, it is possible for a user program to gain control at that point and do something other than to display the registers and return to the Monitor command processor. Such a program must be sure to clear the $04 bit in the P-reg on return. During RESET interrupt handling for power-on, this vector is initialized to point at Autostart Monitor label OLDBRK, which routine does the same thing as was done in Old Monitor. That is, it does a JSR to INSDS1 to display the disassembled instruction at the location indicated by PCL- PCH, a JSR to RGDSP1 to display the register contents, and a JMP to MON to complete the handling of the interrupt. Note: after DOS 3.2 has destroyed page 3 during the bootstrap operation, it restores this vector to point to $FA59, OLDBRK.

ADDRESS TABLE

Function	Hex Addr	+Dec Addr	−Dec Addr	Monitor Label	Registers Destroyed
Disassemble the instruction at (PCL,H), print thru COUT.	F8D0	63696	-1840	INSTDSP	A,X,Y
Display registers thru COUT from save area, after carriage return.	FAD7	64215	-1321	REGDSP	A,X
Display registers thru COUT from save area.	FADA	64218	-1318	RGDSP1	A,X
Save 6502 regs at $45-49.	FF4A	65354	-182	SAVE	A,X
Save A-reg at ACC $45. &					
Save X-reg at XREG $46. &	FF4C	65356	-180	SAV1	
Save Y-reg at YREG $47. &	FF4E	65358	-178		
Save P-reg at STATUS $48. &	FF50	65360	-176		
Save S-reg at SPNT $49. &	FF54	65364	-172		
Clear 6502 decimal mode (set hex).					
Clear 6502 decimal mode (set hex) &	FF65	65381	-155	MON	
Sound bell. &	FF66	65382	-154		
Monitor Command Processor Entry. Set "*" as prompt character.	FF69	65385	-151	MONZ	
AUTOSTART IRQ/BRK HANDLING					
Determine whether interrupt was IRQ or BRK, transfer control accordingly.	FA40	64064	-1472	IRQ	A
Handle BRK interrupt: Restore P-reg from stack. Save registers (SAV1) X,Y,P,S. Move interrupt location from stack to PCL,H. JMP (BRKV) to possibly user specified routine (normally to OLDBRK, below).	FA4C	64076	-1460	BREAK	A,X,Y
Default BRK interrupt handler completion routine Display instruction (2 bytes past), Display registers, JMP to MON.	FA59	64089	-1447	OLDBRK	A,X,Y
OLD MONITOR IRQ/BRK HANDLING					
Determine whether interrupt was IRQ or BRK, transfer control accordingly.	FA86	64134	-1402	IRQ	A
Handle BRK interrupt: Save registers, Display instruction (2 bytes past), Display registers, JMP to MON.	FA92	64146	-1390	BREAK	A,X,Y

PCL,H	58,59	$3A,3B	YREG	71	$47
ACC	69	$45	YSAV	52	$34
XREG	70	$46	STATUS	72	$48

CHAPTER 4

MISCELLANY

MACHINE LANGUAGE DEVELOPMENT AIDS

There are many routines in the Monitor which can be helpful when developing machine language programs. Some of these are routines to be used in the finished program, like the Monitor MOVE routine. Others in this list are general, special, or very special screen output routines, and some data manipulation routines.

ADDRESS TABLE

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Write byte in A to screen at CV,CH.	FDED	65005	-531	COUT	?A
Print carriage return thru COUT.	FD8E	64910	-626	CROUT	A
Print three blanks thru COUT.	F948	63816	-1720	PRBLNK	A,X
Print (X) blanks thru COUT.	F94A	63818	-1718	PRBL2	A,X
Print character in A followed by (X)-1 blanks.	F94C	63820	-1716	PRBL3	A,X
Print BELL code thru COUT.	FF3A	65338	-198	BELL	A
Print "ERR" and BELL thru COUT.	FF2D	65325	-211	PRERR	A
Print low nibble of A as hex char,	FDE3	64995	-541	PRHEX	A
Print A-reg as 2 hex nibbles.	FDDA	64986	-550	PRBYTE	A
Print hex of Y,X regs.	F940	63808	-1728	PRNTYX	A
Print hex of A,X regs.	F941	63809	-1727	PRNTAX	A
Print hex of X-reg.	F944	63812	-1724	PRNTX	A
Print CR, then hex of Y,X regs, then minus sign (or dash).	FD96	64918	-618	PRYX2	A,Y
Print hex of Y,X regs, then dash.	FD99	64921	-615		A,Y
Print CR, hex of A1H,A1L, and dash.	FD92	64914	-622	PRA1	A,X,Y
Print memory as hex with preceeding address from mmmm to mmm7 where mmmm is initial content of A1L,H.	FDA3	64931	-605	XAM8	A (Y=0)
Print memory as hex from (A1L,H) thru (A2L,H).	FDB3	64947	-589	XAM	A (Y=0)
Save A,X,Y,P,S regs at $45-49.	FF4A	65354	-182	SAVE	A,X
Display registers with names from $45-49 as SAVEd, with preceeding carriage return.	FAD7	64215	-1321	REGDSP	A,X
Display regs as above without CR.	FADA	64218	-1318	RGDSP1	A,X
Restore regs A,X,Y,P not S from $45	FF3F	65343	-193	RESTORE	A,X,Y,P
Monitor Command Processor GO entry.	FEB6	65206	-330	GO	A,X,Y,P
Set PCL,H from A1L,H if entered. &					
Call RESTORE, set all regs but S.&	FEB9	65209	-327		
Jump via PCL,H.	FEBC	65212	-324		
Move memory contents to (A4L,H) from (A1L,H) thru (A2L,H).	FE2C	65068	-468	MOVE	A (Y=0)

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Compare memory contents (A4L,H) to (A1L,H) thru (A2L,H), print differences thru COUT.	FE36	65Ø78	-458	VFY	A (Y=Ø)
Increment A4L,H ($42-43). &	FCB4	64692	-844	NXTA4	A
Increment A1L,H ($3C-3D), set Carry if A2L,H less than A1L,H.	FCBA	64698	-838	NXTA1	A
Set GBASL,H for line (A).	F847	63559	-1977	GBASCALC	A
Clear A-reg to a nibble, leaving in low nibble entry low nibble if entry carry clear, high nibble if entry carry set.	F879	636Ø9	-1927	SCRN2	A
Disassemble the instruction at (PCL,H), print thru COUT.	F8DØ	63696	-184Ø	INSTDSP	A,X,Y
Compute (PCL,H) + (LENGTH), leave results in A,Y. Decimal Mode Flag must be clear before calling PCADJ.	F953	63827	-17Ø9	PCADJ	A,X,Y
Read paddle (X) into (Y-reg).	FB1E	64286	-125Ø	PREAD	A,Y
Wait .Ø1 seconds, then sound bell.	FBDD	64477	-1Ø59		A,Y
Load Y=192 for .1 sec of bell. &	FBE2	64482	-1Ø54		A,Y
Toggle speaker at 1 KHZ for number of cycles in Y-reg.	FBE4	64484	-1Ø52	BELL2	A,Y
Place character in screen refresh memory if not control character. If known control character, do it. If unknown control character, RTS.	FBFD	645Ø9	-1Ø27	VIDOUT	A,Y
Clear window to blank, set cursor to top left corner.	FC58	646ØØ	-936	HOME	A,Y
Load Ø into Y, then print dash.	FD9C	64924	-612		
Print dash thru COUT.	FD9E	64926	-61Ø		
Character print to screen output routine entry - normal for CSWL. Print character to screen with appropriate actions on controls and control characters. If (A)<$AØ goto COUTZ, bypass inverse video mask.	FDFØ	65ØØ8	-528	COUT1	?A
Monitor entry on RESET key pressed or Power on. Call SETNORM - white on black. &	FF59	65369	-167	RESET	
Clear 65Ø2 decimal mode (set hex). &	FF65	65381	-155	MON	
Sound bell &	FF66	65382	-154		
Monitor Command Processor Entry. Set "*" as prompt character &	FF69	65385	-151	MONZ	
Set (a) as prompt character &	FF6B	65387	-149		
Monitor Command Processor command parsing routine; save hex digits in A2L,H, return with command (first non-hex) in A-reg, Y-reg set for next character.	FFA7	65447	-89	GETNUM	

A1L,H	6Ø,61	$3C,3D	A4L,H	66,67	$42,43	YREG	71	$47
A2L,H	62,63	$3E,3F	PCL,H	58,59	$3A,3B	XREG	7Ø	$46
A3L,H	64,65	$4Ø,41	ACC	69	$45			

LORES PLOTTING

In standard (or low resolution) plotting mode, the graphic area of the screen is 4Ø points wide and either 4Ø points high with 4 lines of text below or 48 lines high. The X coordinate is horizontal and the Y coordinate is vertical. The same memory area is used for low resolution plotting as is used for text output to the screen. However, in the graphics mode, each character position contains information for two plot points, one immediately above the other. Thus, 2Ø text lines are used to display 4Ø graphics lines in the mixed mode, and 24 text lines are used to display 48 graphics lines in the full screen mode.

There are four bits allocated for each point, by means of which the point may be displayed in any of 16 colors.

The Monitor contains routines supporting the following functions:

Set display mode to mixed graphics and text.

Clear the graphics part of the screen (in whole or in limited part).

Set a color control byte to be used for each plot point established until another color is selected.

Plot a single point at an indicated vertical/horizontal position.

Plot a horizontal line from one vertical/horizontal point to a vertical value.

Plot a vertical line from one vertical/horizontal point to a vertical value.

Return to requesting program the color value of the point at a specified coordinate.

There are limitations on some of these functions which may not always be desirable. For example, using the entry point which sets mixed graphics and text includes clearing the graphics part of the screen, setting the Scroll Window to be the entire remainder of the screen, and moving the cursor (straight down from current position) to the bottom line of the screen. In addition, there is no Monitor entry point for setting full screen graphics mode. However, the display mode controls are easily set in any desired fashion merely by poking or storing into the appropriate memory locations, so this is certainly no major problem.

Various page zero locations are used for low resolution graphics mode.

PAGE ZERO FIELDS

Routine	Dec Addr.	Hex Addr.	Description
GBASL,H	38-39	$26-27	Is set by the GBASCALC routine to the memory address of the plotting line specified.
COLOR	48	$3Ø	contains the selected color value in both high and low nibbles of the byte.
MASK	46	$2E	is used internally by the plot routines as $FØ or $ØF to set either the high or low nibble of the receiving byte depending on whether the graphics line is the top or bottom of the two displayed from that "text" line.
H2	44	$2C	is the right end point for horizontal line drawing.
V2	45	$2D	is the bottom end point for vertical line drawing.

ADDRESS TABLE

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Plot a point at line (A) col. (Y) leaving GBASL,H and MASK set.	F8ØØ	63488	-2Ø48	PLOT	A
Plot a point, line per GBASL,H and MASK, col. in Y.	F8ØE	635Ø2	-2Ø34	PLOT1	A
Draw horizontal line at (A) from (Y) thru (H2), left to right.	F819	63513	-2Ø23	HLINE	A,Y
Draw horizontal line at line indicated by GBASL,H. MASK from (Y) thru (H2).	F81C	63516	-2Ø2Ø	HLINE1	A,Y
Plot vertical line at (Y) from (A) thru (V2).	F828	63528	-2ØØ8	VLINE	A
Plot vertical line at (Y) from (A)+1+carry thru (V2).	F826	63526	-2Ø1Ø	VLINEZ	A
Plot vertical line at (Y) from (A)+1 thru (V2).	F82D	63533	-2ØØ3		A
Clear full (48 lines) screen.	F832	63538	-1998	CLRSCR	A,Y
Clear graphics area (4Ø lines).	F836	63542	-1994	CLRTOP	A,Y
Clear graphics partial from line Ø thru (Y), 4Ø col. wide.	F838	63544	-1992	CLRSC2	A,Y
Clear graphics partial from line Ø to (V2) 4Ø col. wide.	F83A	63546	-199Ø		A,Y
Clear graphics partial, top left lines Ø thru (V2),col. Ø thru (Y).	F83C	63548	-1998	CLRSC3	A,Y

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Set LORES screen to COLOR from top left corner to (Y),(V2). Entry A-reg must be Ø. Entry Y-reg = right column to set. Set V2 to last line to set.	F84Ø	63552	-1984		A,Y
Set COLOR for following points to (A).	F864	63588	-1948	SETCOL	A
Change COLOR to (COLOR)+3.	F85F	63583	-1953	NXTCOL	A
Load to A color of point (A),(Y).	F871	636Ø1	-1935	SCRN	A
Set GBASL,H from A. (A)=line/2.	F847	63559	-1977	GBASCALC	A
Set Color Graphics display mode and following are also done;	FB4Ø	6432Ø	-1216	SETGR	A,Y
Set graphics mode to Mixed.	& FB43	64323	-1213		A,Y
Clear graphics part of screen.	& FB46	64326	-121Ø		A,Y
Load $14 to A for WNDTOP.	& FB49	64329	-12Ø7		A
Store A to WNDTOP.	& FB4B	64331	-12Ø5	SETWND	A
Load Ø to A for WNDLFT.	& FB4D	64333	-12Ø3		A
Store A to WNDLFT.	& FB4F	64335	-12Ø1		A
Load $28 to A for WNDWDTH.	& FB51	64337	-1199		A
Store A to WNDWDTH.	& FB53	64339	-1197		A
Load $18 to A for WNDBTM.	& FB55	64341	-1195		A
Store A to WNDBTM.	& FB57	64343	-1193		A
Load $17 to A for CV. Go to TABV to set BASL,H.	& FB59	64345	-1191		A

DATA MANUPILATION FUNCTIONS

There are a number of routines in the Monitor which may be called by user programs to perform often needed tasks. The routines described in this section are miscellaneous routines which move data from place to place or convert the form of information provided to the routines. Note that some of these routines are in both the Old Monitor and the Autostart Monitor while other routines are in only one or the other. Three address tables are provided; one for both Monitors, one for the Old Monitor, and one for the Autostart Monitor.

ROUTINES

Memory to Memory Move

This routine is used by the Monitor "M" command. As the Command Interpreter scans the keyboard input, fields A1, A2, and A4 are loaded. When the Command Interpreter encounters the "M" it calls label MOVE, as indicated in the table. The contents of memory from locations (A1) thru (A2) are moved to memory beginning at location (A4). See the sample program in the section "Secondary Display Area Ways and Means" for use of MOVE from BASIC, with the assistance of the Monitor GO routine for setting registers on the way in.

Jump to Address with Registers Loaded

The routine in the Monitor which responds to the "G" command uses some Monitor routines from BASIC or APPLESOFT in that the registers are loaded from the save area and then control is transferred to the location specified in PCL,H. Thus, a BASIC program can set up the destination address and register contents, and then CALL -468 to have the requested routine entered. This is used in sample programs in this section and in the section on "Secondary Display Areas".

Increment Address Fields

The Monitor Move routine described above is a sample caller of the NXTA4 and NXTA1 routines. When NXTA4 is called, it increments the two byte field A4L,H and then falls into label NXTA1. The routine at NXTA1 increments the two byte field at A1L,H, and then compares that field to the two byte field A2L,H before returning to the calling program. On return to the calling program, the Carry status bit is clear if (A1L,H) is less than or equal to (A2L,H). Carry is set if (A1L,H) is greater than (A2L,H).

Save 6502 Registers

The SAVE routine is used by various other Monitor routines to store the 65Ø2 registers in page zero locations $45-$49. This routine may be called by user program under certain conditions - namely, that neither the Monitor nor any other program will be calling SAVE at the same time. In the Old Monitor SAVE and RESTORE are used in support of Monitor commands S and T, single step and instruction trace. In both Monitors, the SAVE routine is called on a BRK interrupt at entry point SAV1 as the A-reg is stored at $45 on entry into IRQ interrupt processing.

Restore 6502 Registers

The routine at label RESTORE is the inverse of the SAVE routine, except that the S-reg is not loaded. In the Old Monitor, RESTORE is utilized by instruction step and trace routines before controlled execution of each traced instruction. In both Monitors, the registers are loaded by RESTORE in execution of the Monitor G command before transferring control to the operator-indicated location.

Multiply Two Byte Fields

The MUL and MULPM routines multiply two byte fields to give a four byte product. They exist only in the Old Monitor. If a program (such as an assembler) calls MULPM at FB6Ø, and it is executed with the Autostart Monitor in the machine, the result is that on each call the screen will be cleared and "APPLE II" will be written on the top line.

Multiply Routine

Note in the following that the data fields for multiply and divide are in the same format as other multiple byte numbers in the Apple: lowest memory address is least significant byte.

Set Multiplier in	$55,54 (MSB,LSB)
Set Multiplicand in	$51,5Ø (MSB,LSB)
Should be zero - see note	$53,52

Call/JSR FB6Ø or FB63 (-1184 or -1181) (MULPM or MUL) depending on sign conventions or requirements.

The result, in order of most significant to least, is in $53, $52, $51, $5Ø. this result is positive. If one of the two input factors (but not both) was negative, then SIGN (at $2F) contains an $Ø1 bit, indicating that the result should be complemented by the user program before further use.

NOTE: The table of values above indicates that $53,52 should be set to zero before calling multiply. If this is not done, then the initial contents of this field will be added to the result. For example, if a table has an origin of $84ØØ with 7 byte long entries, the address of entry 8 can be determined by entering the multiply with $84ØØ in $51,52 and the 8 and 7 in position for the multiply.

Examples:

Called	Inputs				Outputs				
Routine	$51	$5Ø	$55	$54	$53	$52	$51	$5Ø	$2F
MULPM	ØØ	Ø1	ØØ	Ø1	ØØ	ØØ	ØØ	Ø1	ØØ
	ØØ	Ø1	Ø1	ØØ	ØØ	ØØ	Ø1	ØØ	ØØ
	Ø4	ØØ	Ø8	ØØ	ØØ	2Ø	ØØ	ØØ	ØØ
	FC	ØØ	Ø8	ØØ	ØØ	2Ø	ØØ	ØØ	Ø1
	FC	ØØ	F8	ØØ	ØØ	2Ø	ØØ	ØØ	Ø2
	7F	FF	7F	FF	3F	FF	ØØ	Ø1	ØØ
	8Ø	ØØ	Ø2	ØØ	Ø1	ØØ	ØØ	ØØ	Ø1
	8Ø	ØØ	8Ø	ØØ	4Ø	ØØ	ØØ	ØØ	Ø2
MUL	ØØ	Ø1	ØØ	Ø1	ØØ	ØØ	ØØ	Ø1	
	ØØ	Ø1	Ø1	ØØ	ØØ	ØØ	Ø1	ØØ	
	Ø4	ØØ	Ø8	ØØ	ØØ	2Ø	ØØ	ØØ	
	FC	ØØ	Ø8	ØØ	Ø7	EØ	ØØ	ØØ	
	FC	ØØ	F8	ØØ	F4	2Ø	ØØ	ØØ	
	ØØ	FC	ØØ	F8	ØØ	ØØ	F4	2Ø	
	8Ø	ØØ	Ø2	ØØ	Ø1	ØØ	ØØ	ØØ	
	8Ø	ØØ	8Ø	ØØ	4Ø	ØØ	ØØ	ØØ	
	12	34	56	78	Ø6	26	ØØ	6Ø	

Divide Four Byte Dividend by Two Byte Divisor

This routine divides a four byte dividend by a two bit divisor, giving a two byte quotient and a two byte remainder. It is available only in the Old Monitor. This routine accomplishes the division of the number in bytes $53,52,51,5Ø by the number in bytes $55,54, leaving the quotient in $51,5Ø and the remainder in $53,52 (most significant to least significant).

If the contents of $53,52 is larger than the contents of $55,54, then the result will not fit in the quotient bytes - overflow is the result. The calling program must not let this happen.

With regards to scaling, looking at the four byte dividend as an integer value and the divisor in $55,54 as an integer, the quotient and remainder fields are also integers.

Sign can be a problem if the DIVPM entry point is used. The sign bit of the dividend is the $8Ø bit of byte $51. If the intended divide is two bytes (with $53,52 cleared before divide) then signed fields division is supported, with the sign bit being the LSB of $2F. If the call is to DIVPM, and if $2F contains $Ø1, then complement the results before using them.

When using unsigned divide, entry point DIV, then the divide is 32 bit field by 16 bit field with 16 bit results.

Examples:

Called Routine	Inputs Dividend	Inputs Divisor	Outputs Quotient	Outputs Remainder	Sign
	$53 52 51 5Ø	$55 54	$51 5Ø	$53 52	$2F
DIVPM	ØØ 4Ø ØØ ØØ	Ø8 ØØ	Ø8 ØØ	ØØ ØØ	ØØ
[$FB81]	ØØ ØØ ØØ Ø8	ØØ Ø4	ØØ Ø2	ØØ ØØ	ØØ
[64385]	ØØ Ø1 ØØ ØØ	ØØ Ø2	8Ø ØØ	ØØ ØØ	ØØ
[-1151]	ØØ ØØ ØØ Ø3	ØØ Ø2	ØØ Ø1	ØØ Ø1	ØØ
	ØØ ØØ 3Ø ØØ	Ø2 ØØ	ØØ 18	ØØ ØØ	ØØ
	ØØ ØØ 3Ø ØØ	2Ø ØØ	ØØ Ø1	1Ø ØØ	ØØ
	ØØ ØØ 33 33	ØØ 22	Ø1 81	ØØ 11	ØØ
	ØØ 1Ø 4Ø ØØ	Ø4 ØØ	Ø4 1Ø	ØØ ØØ	ØØ
	ØØ 2Ø 8Ø ØØ	Ø8 ØØ	Ø4 1Ø	ØØ ØØ	Ø1
	ØØ 2Ø $\overline{82}$ ØØ	$\overline{Ø8}$ ØØ	Ø4 ØF	Ø6 ØØ	$\overline{Ø1}$
	ØØ 1Ø $\overline{41}$ ØØ	$\overline{Ø4}$ ØØ	Ø4 1Ø	Ø1 ØØ	$\overline{ØØ}$
DIV					
[$FB84]	ØØ 8Ø ØØ ØØ	8Ø ØØ	Ø1 ØØ	ØØ ØØ	
[64388]	ØØ ØØ 8Ø ØØ	Ø8 ØØ	ØØ 1Ø	ØØ ØØ	
[-1148]					

Establish a RESET Vector

The Autostart Monitor supports an address vector for completion of handling a RESET interrupt. It is called the Soft Entry vector as it is designed to allow resumption of processing after a RESET. This vector is in page three. It contains the address to which control is to be transferred after the screen, keyboard, and other basic Apple hardware items have been set to their "initial" states. For example, the display hardware is set to display primary area text, and the Scroll Window full screen values are set.

After such initialization is performed, locations $Ø3F3 and $Ø3F4 are tested against one another to determine whether the vector in $Ø3F2-$Ø3F3 is to be considered valid. If so, control is transferred to ($Ø3F2-Ø3F3). Normally, this results in transfer of control to $EØØ3 to accomplish the result of entry to the Monitor of a control-C, re-entry into BASIC or APPLESOFT. During the bootstrap operation, DOS installs its own restart point in this vector. And, of course, you may wish to set some other value in this vector, such as that which will cause the Monitor (with asterisk prompt) to be called, as was the normal case with the Old Monitor. To set a different value in that vector, POKE or store the desired value in $Ø3F2-$Ø3F3 and then CALL or JSR to SETPWRC ($FB6F or -1169) to have the Monitor set $Ø3F4 appropriately.

Convert Hex Characters to Value for Use

Programmer utility programs often need input of address or data in hex rather than in decimal. The Monitor also uses input in hex, and therefore has a way of converting input hex characters to a value in a field. The GETNUM routine in the Monitor converts characters from the keyboard input area ($Ø2ØØ-$Ø2FF) to hex stored in A2L,H and conditionally in A1L,H and A3L,H.

The GETNUM routine converts characters in the $Ø2ØØ area beginning at $Ø2ØØ+(Y-reg) and continuing until a character is found which is not a hex digit (not Ø-9 or A-F). The result in A2L,H (and A1L,H and A3L,H if (MODE) = Ø) is the last four hex digits in the string converted if the string is more than four hex digits. If the string is fewer than four hex digits the result field contains the value right adjusted with leading zeroes. A sample program is provided at the end of this section showing use of GETNUM from APPLESOFT.

Disassemble an Instruction

The Apple II Monitor contains a disassembler by means of which one can display a portion of a machine language program in mnemonics instead of just hex. At label LIST ($FE5E) is the routine to which control is passed when the Monitor command "L" is used. This routine sets a

counter to 2Ø, and then calls the single instruction disassembler 2Ø times, with appropriate adjustment of the instruction pointer PCL,H. This routine can be used as an example of how to use the locations in the address table with labels INSTDSP and PCADJ.

The routine at INSTDSP uses the INSDS1 routine to set the zero page locations FORMAT and LENGTH appropriately for the instruction at (PCL,H). INSDS1 also prints to the screen the contents of PCL,H, the address of the instruction to be disassembled. On return from INSDS1, the INSTDSP routine controls the printing of the rest of the disassembly line.

Note that PCL,H is not altered by disassembly of the instruction. Thus, it must be "maintained" by the program which calls INSTDSP. This is accomplished by calling the PCADJ routine, which returns the new values to the calling program, to store into PCL and PCH in the A-reg and Y-reg, respectively, having computed the new value from PCL and PCH and LENGTH (set by INSDS1).

ADDRESS TABLE

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
OLD MONITOR ONLY					
Multiply signed fields leaving sign in LSB of SIGN.	FB6Ø	64352	-1184	MULPM	A,X,Y
Multiply fields unsigned, (51,5Ø) * (55,54) = (53,52,51,5Ø).	FB63	64355	-1181	MUL	A,X,Y
Divide signed fields leaving sign in SIGN LSB (from 51,55).	FB81	64385	-1151	DIVPM	A,X,Y
Divide unsigned fields (53,52,51,5Ø)/(55,54)=(51,5Ø).	FB84	64388	-1148	DIV	A,X,Y
Set absolute values for ACL,H and AUXL,H leaving resulting sign in LSB of SIGN (called by MULPM and DIVPM).	FBA4	6442Ø	-1116	MD1	A,X,Y
AUTOSTART MONITOR ONLY					
Set validity of RESET vector.	FB6F	64367	-1169	SETPWRC	A
BOTH OLD AND AUTOSTART MONITORS					
Monitor Command Processor GO entry.	FEB6	652Ø6	-33Ø	GO	A,X,Y,P
Set PCL,H from A1L,H if entered. & Call RESTORE, set all regs but S. &	FEB9	652Ø9	-327		
Jump via PCL,H.	FEBC	65212	-324		

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Move bytes in memory to (A4L,H) from (A1L,H) thru (A2L,H). Note: Y-reg must be zero on entry.	FE2C	65Ø68	-468	MOVE	A
Increment pointer A4L,H &	FCB4	64692	-844	NXTA4	A
Increment pointer A1L,H with set of carry if resulting (A1L,H) is greater than (A2L,H).	FCBA	64698	-838	NXTA1	A
Save 65Ø2 regs A,X,Y,P,S at \$45-\$49.	FF4A	65354	-182	SAVE	A,X
Restore 65Ø2 regs A,X,Y,P from \$45-\$48.	FF3F	65343	-193	RESTORE	A,X,Y,P
Convert hex characters from \$2ØØ,Y to value in A2L,H (and A1L,H and A3L,H if (MODE)=Ø).	FFA7	65447	-89	GETNUM	A,X,Y
Disassemble one instruction with display thru COUT.	F8DØ	63696	-184Ø	INSTDSP	A,X,Y
Compute new PCL,H after disassembly or trace or step - return results in A,Y regs for (PCL,H).	F953	63827	-17Ø9	PCADJ	A,X,Y

APPLESOFT SAMPLE DATA MANIPULATION PROGRAM

```
1Ø             REM  DATA MANIPULATION FUNCTIONS
2Ø             REM   SAMPLE PROGRAM
3Ø             REM  MEMORY DUMP
4Ø             REM   OF HEX AREA INDICATED.
5Ø    GOTO 1ØØØ:        REM  BYPASS SUBROUTINES
2ØØ            REM   CALL GETNUM ROUTINE VIA GO ROUTINE
21Ø   POKE 58,167:      REM  PCL=$A7
22Ø   POKE 59,255:      REM  PCH=$FF
23Ø   SI$ = AD$ + " ":  REM  BUILD STRING TO STORE
24Ø   FOR I = 1 TO  LEN (SI$)      REM:  STORE STRING IN INPUT BUFFER
25Ø   CC$ =  MID$ (SI$,I,1)        REM:                   ''
26Ø   CC% =  ASC (CC$) + 128       REM:                   ''
27Ø   POKE 512 + I,CC%
28Ø   NEXT
29Ø   POKE 71,1:        REM  SET YREG TO START AT LOCATION 513
3ØØ   POKE 49,Ø:        REM  CLEAR MODE BYTE
31Ø   CALL  - 327:      REM  GO PROCESSOR
32Ø   ST =  PEEK (62) + 256 *  PEEK (63):   REM  ST=START ADDRESS($A2)
33Ø   IF ST > 32767 THEN ST = ST - 65536    REM  TWO'S COMPLEMENT
                                                 ADDRESS IF >= $8ØØØ
34Ø   RETURN
```

```
600          REM  DISPLAY HEX CONTENTS
610   SH% = ST / 256      REM GET HI ADDRESS BYTE
620   SL% = ST - SH% * 256:  REM  GET LO ADDRESS BYTE
630   IF SH% < 0 THEN SH% + 256:  REM  GET 2'S COMP IF NECESSARY
640   POKE 60,SL%:POKE 61,SH%
650   RM% = SL% - ( INT (SL% / 8 )) * 8  REM  RM% = MOD 8 OF LO BYTE
660   IF RM% THEN CALL -622
670   POKE 71,0:  REM  SET "Y" REG TO ZERO
680   POKE 58,163:    REM  PCL = $A3
690   POKE 59,253:    REM  PCH = $FD
700   CALL  - 327:    REM  CLEAR "Y" REG & $FDA3G
710   POKE 36,29:  PRINT "I ";:  REM  SEPARATES HEX FROM ASCII
720          REM  DISPLAY ASCII CHARACTER CONTENTS
730   SE = ST + 7 - RM%   REM  SEPARATES HEX FROM ASCII
740   FOR I = ST TO SE    REM  PRINT ASCII CONTENTS
750   CX =  PEEK (I): IF CX < 128 THEN CX = CX + 128
760   CX$ =  CHR$ (CX): IF CX < 160 THEN CX$ = "?"
770   PRINT CX$;
780   NEXT
790   RETURN
1000         REM  PROGRAM START
1010  PRINT "HEX DISPLAY"
1020  INPUT "ENTER ADDRESS ";AD$
1030  IF AD$ = "END" THEN  END
1040  IF  LEN (AD$) = 0 THEN 1100:REM CONTINUE WITH NEXT AVAILABLE
                                      ADDRESS
1050  GOSUB 200:
1080  FOR J = 1 TO 16:    REM  PRINT 16 LINES
1090  GOSUB 600
1100  ST = ST + 8 - RM%
1110  NEXT
1120  PRINT
1130  GOTO 1020
```

MONITOR COMMAND PROCESSOR

The Monitor Command Processor is that part of the Monitor which responds to commands entered with the "*" prompt character. These commands include data movement from one location to another, cassette tape reading and writing, instruction disassembly, and others described in the Reference Manual. The Reference Manual contains a complete description of use of these commands. This section of this manual describes calling some of the routines from a user program instead of from the keyboard, and jumping into the Monitor with no return to the user program.

ENTERING THE MONITOR COMMAND PROCESSOR

The Monitor Command Processor is that part of the Monitor which reads keyboard input with the asterisk prompt character and performs the requested service. "Entering" the Command Processor implies turning over control of the machine to the Monitor Mode. When the RESET key is pressed with the Old Monitor in the Apple the computer is placed in

Monitor Mode. When the RESET key is pressed with the Autostart Monitor in the machine, the computer generally goes into BASIC or APPLESOFT. With the Autostart Monitor the only way to get into Monitor Mode is to CALL one of these entry points (generally CALL - 151).

In this mode, data may be moved in memory using the Monitor Move command. Blocks can be read from tape via the cassette tape data transfer commands. Or any of the other Monitor commands may be used. However, having entered Monitor Mode, the Monitor Command Processor is reading the commands from the keyboard and then acting upon them.

There are a number of entry points indicated in the address table for "entering" the Monitor Command Processor. Please note that once the Monitor is jumped to at the specified point, all of the initialization described after that entry point is also performed. This is implied by the "&" at the end of each function description.

CALLING THE MONITOR COMMAND PROCESSOR

"Calling" the Monitor Command Processor implies that return will take place to the calling program. However, the driver part of the Monitor Command Processor is not designed to operate in that fashion, so a short machine language program is required to allow exit back to the calling program. A sample program is provided at the end of this section indicating the required setup. In the sample, the three byte machine language routine is placed in page two (at $02FC) but it may be placed anywhere desired. With this program, Monitor calls from BASIC or APPLESOFT are both supported.

A program which CALLs the Command Processor must first store the three byte exit routine somewhere. Then the program can POKE a string of Monitor commands into the input area, beginning at address $0200, the last command of each such string being a Monitor GO command to transfer control to the exit routine. In the sample, the last Monitor command in the string is "02FCG". The function of the exit routine is to pull one return address level (two bytes) off of the stack, and then do an RTS to return to the BASIC, APPLESOFT, or machine language calling program.

ADDRESS TABLE

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Monitor Command Processor, "blank" entry point used for CR.	FE00	65024	-512	BL1	A,X,Y
Monitor Command Processor, "blank" command entry point.	FE04	65028	-508	BLANK	A,X,Y
Monitor Command Processor, Store routine.	FE0B	65035	-501	STOR	A
Monitor Command Processor, set MODE for colon, period, plus, or minus.	FE18	65048	-488	SETMODE	A,Y

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Store appropriate value to MODE, entered from BLANK also.	FE1D	65Ø53	-483	SETMDZ	none
Monitor Command Processor routine for less than (<) command.	FE2Ø	65Ø56	-48Ø	LT	A,X
Monitor Command Processor MOVE routine. (A1L,H) thru (A2L,H) is moved to (A4L,H) thru whatever.	FE2C	65Ø68	-468	MOVE	A (Y=Ø)
Monitor Command Processor VERIFY routine. (A1L,H) thru (A2L,H) is compared to (A4L,H) thru whatever with differences printed thru COUT.	FE36	65Ø78	-458	VFY	A (Y=Ø)
Monitor Command Processor LIST (disassembler) routine: list 2Ø instructions thru COUT.	FE5E	65118	-418	LIST	A,X,Y
Set INVFLG to $3F = inverse video.	FE8Ø	65152	-384	SETINV	Y
Set INVFLG to $FF = normal video.	FE84	65156	-38Ø	SETNORM	Y
Set INVFLG from Y-reg.	FE86	65158	-378	SETIFLG	none
Set port Ø (keyboard) for input.	FE89	65161	-375	SETKBD	A,X,Y
Set port (A) for input.	FE8B	65163	-373	INPORT	A,X,Y
Set port (A2L) for input.	FE8D	65165	-371	INPRT	A,X,Y
Set port Ø (screen) for output.	FE93	65171	-365	SETVID	A,X,Y
Set port (A) for output.	FE95	65173	-363	OUTPORT	A,X,Y
Set port (A2L) for output.	FE97	65175	-361	OUTPRT	A,X,Y
Monitor Command Processor GO entry.	FEB6	652Ø6	-33Ø	GO	A,X,Y,P
Set PCL,H from A1L,H if entered. &					
Call RESTORE, set all regs but S.&	FEB9	652Ø9	-327		
Jump via PCL,H.	FEBC	65212	-324		
Monitor Command Processor Display Register contents.	FEBF	65215	-321	REGZ	
Monitor Command Processor Carriage Return entry. First, simulate entry of blank. Then POP 2 from stack and goto Monitor Command Processor at MONZ.	FEF6	6527Ø	-266	CRMON	
Restore registers from $45-49:	FF3F	65343	-193	RESTORE	
Load STATUS and push to stack. &					
Load A from ACC. &	FF42	65346	-19Ø		
Load X from XREG. &	FF44	65348	-188	RESTR1	
Load Y from YREG. &	FF46	6535Ø	-186		
Load P from stack (PLP) and RTS.	FF48	65352	-184		
Save 65Ø2 regs at $45-49.	FF4A	65354	-182	SAVE	
Save A-reg at ACC $45. &					
Save X-reg at XREG $46. &	FF4C	65356	-18Ø	SAV1	
Save Y-reg at YREG $47. &	FF4E	65358	-178		
Save P-reg at STATUS $48. &	FF5Ø	6536Ø	-176		
Save S-reg at SPNT $49. &	FF54	65364	-172		
Clear 65Ø2 decimal mode (set hex).					

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
Monitor entry on RESET key pressed or Power on.	FF59	65369	-167	RESET	
Call SETNORM - white on black. &					
Call INIT - Text + full scroll. &	FF5C	65372	-164		
Call SETVID - screen as output. &	FF5F	65375	-161		
Call SETKBD - keyboard = input. &	FF62	65378	-158		
Clear 65Ø2 decimal mode (set hex).&	FF65	65381	-155	MON	
Sound bell. &	FF66	65382	-154		
Monitor Command Processor Entry.	FF69	65385	-151	MONZ	
Set "*" as prompt character. &					
Set (A) as prompt character. &	FF6B	65387	-149		
Call GETLNZ to read command line. &	FF6D	65389	-147		
Clear MODE before scanning line. &	FF7Ø	65392	-144		
Pick up one command: Call GETNUM to scan input line, saving hex digits in A2L,H, and returning with non-hex in A-reg. Save Y at YSAV - current place in command line.	FF73	65395	-141	NXTITM	
Call routine indicated by non-hex returned by GETNUM.	FF82	6541Ø	-126		
On return from Monitor Command Service routine, reload Y from YSAV and goto NXTITM to process next command in the line, if any.	FF85	65413	-123		
Monitor Command Processor command parsing routine; save hex digits in A2L,H, return with command (first non-hex) in A-reg, Y-reg set for next character.	FFA7	65447	-89	GETNUM	
Call routine indicated by command character: Push address $FExx onto stack. Pass (MODE) to called routine in A-reg. Clear MODE before call. Call selected routine by RTS.	FFBE	6547Ø	-66	TOSUB	
Clear MODE byte between commands.	FFC7	65479	-57	ZMODE	
OLD MONITOR ONLY					
Execute instruction at (PCL,H), with display of instruction and result registers.	FA43	64Ø67	-1469	STEP	
Monitor Command Processor TRACE instructions routine.	FEC2	65218	-318	TRACE	
Monitor STEP one instruction.	FEC4	6522Ø	-316	STEPZ	

A1L,H	6Ø,61	$3C,3D	PCL,H	58,59	$3A,3B
A2L,H	62,63	$3E,3F	ACC	69	$45
A3L,H	64,65	$4Ø,41	XREG	7Ø	$46
A4L,H	66,67	$42,43	YREG	71	$47
YSAV	52	$34			

APPLESOFT SAMPLE PROGRAM

```
1           REM  MONITOR COMMAND PROCESSOR SAMPLE PROGRAM
1Ø AA$ = "2FC:68 68 6Ø N 2FCG ": REM SET UP RETURN ROUTINE @2FC
11    GOSUB 1ØØØ:       REM  MOVE COMMAND TO KEYBOARD INPUT AREA

1ØØ                   REM  RETURN IS SET. NOW CALL
1Ø1                   REM  SOME MONITOR COMMANDS.
11Ø   AA$ = "F8ØØL 1ØØ.1FF 2FCG "
12Ø   CALL  - 936:    REM  CLEAR THE SCREEN
13Ø   GOSUB 1ØØØ:     REM DO DISASSEMBLY, MEMORY DISPLAY, RETURN
14Ø   PRINT : PRINT :
141   PRINT "THATS ALL. "
15Ø   END

1ØØØ  B = 511:  REM FOR LOOP IS 1 TO LIM, SO B=BYTE BEFORE $2ØØ
1ØØ5  LIM =  LEN (AA$)
1Ø1Ø  FOR I = 1 TO LIM
1Ø2Ø  P$ =  MID$ (AA$,I,1)
1Ø5Ø  P =  ASC (P$) + 128
1Ø7Ø  POKE B + I,P
1Ø8Ø  NEXT
1Ø85  CALL  - 144
1Ø9Ø  RETURN
```

SPEAKER USE THROUGH THE MONITOR

There are many ways to use the speaker in the Apple II. One of these ways is to signal program events. The Monitor contains a routine which supports this use by toggling the speaker at 1 khz for .1 second. This is the "beep" heard when the RESET key is pressed or at completion of a tape record read or write.

The Apple II does not contain the only speaker in town. That is, some printers which attach to the Apple II make a sound of some type when presented with the BELL code. On the Apple II keyboard this is the control-G. The character code is $87 or decimal 135. "Printing" this character through COUT will cause the Apple to beep, and will cause a printer "bell" to sound if there is one.

There are two ways for a user program to call the routine in the Monitor which responds to output of $87 by sounding the beep.

> If you intend to sound the bell in the Apple regardless of output device in use, then directly call the routine in the Monitor which produces the sound; CALL -1Ø59 (or CALL 64477), or JSR FBDD expecting destruction of the A- reg and Y-reg.

> If you want to sound the bell of the Apple II if the screen is the print device, or to sound the speaker in the printer, call the entry point in the Monitor which places a $87 in the A- reg and "prints" it through COUT; CALL-198 (or CALL 65338) or JSR FF3A expecting destruction of the A-reg.

ADDRESS TABLE

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Registers Destroyed
If (A)=$87 wait .Ø1 seconds, then sound the "bell". Else, RTS.	FBD9	64473	-1Ø63	BELL1	A,Y
Wait .Ø1 seconds, then sound bell.	FBDD	64477	-1Ø59		A,Y
Load Y = 192 for .1 sec of bell. &	FBE2	64482	-1Ø54		A,Y
Toggle speaker at 1 KHZ for number of cycles in Y-reg.	FBE4	64484	-1Ø52	BELL2	A,Y
Print thru COUT "ERR" and bell code.	FF2D	65325	-211	PRERR	A
Print bell code ($87) thru COUT.	FF3A	65338	-198	BELL	A

CASSETTE TAPE INPUT AND OUTPUT

There are two primary entry points in the Monitor with regard to reading and writing tape. They are READ and WRITE. The requirements for Calling these are described below. There are a number of other routine entry points which are used by the Monitor on bit and byte basis. These are described below to the extent of location in the Monitor and indication of which Apple II programs call them, but the precise timings of instructions between consecutive calls is beyond the scope of this manual.

As you will have found by now, some tape files are composed of one record, and some of two records. For example, LOADing an APPLESOFT or BASIC program results in two beeps, signaling the completions of the reads of two separate records from the tape.

Definitions are in order:

> A tape record is a single contiguous string of bits which is read into or written from memory as a unit. A tape record is a physical entity.

> A file on tape is a series or sequence of one or more records containing data in a logical organization. A file is a logical entity.

An APPLESOFT or BASIC program file consists of two records. For BASIC, the first of these records is two bytes long, and contains the length of the second record. When the Monitor has satisfied BASIC's read of the first record, BASIC uses the record length indicated in that record to determine the start and end points in memory into which the Monitor will read the second record. Each call to READ or WRITE in the Monitor accomplishes only one record input or output.

APPLESOFT programs are also SAVEd as two record sets or files. However, the first record is three bytes long: the first two bytes indicate the length, and the third byte is set to $55 to indicate a normal APPLESOFT II (as differentiated from APPLESOFT I) program.

Some other programs write a longer (but fixed length) first record containing length of the second record of the file, and other information about the file such as date of creation or name of the file.

WRITE

$FECD 65229 -3Ø7

Before entry at this point, set the first byte address in A1L,H ($3C-3D) and the last byte address at A2L,H ($3E-3F). The Monitor will write ten seconds of continuous tone (header) followed by the contents of memory as specified, followed by one byte of checksum (the result of Exclusive OR of all the data bytes written to the tape).

READ

$FEFD 65277 -259

Before entry at this point, place the first byte address into A1L,H ($3C-3D) and the last byte address into A2L,H ($3E-3F). The Monitor reads the data from the tape, storing it into memory in the specified locations, and maintaining a running Exclusive OR result in the zero page field called CHKSUM ($2E). When the last specified memory location has been filled from the tape, the Monitor reads one more byte and compares it with the contents of CHKSUM. If equal, the Monitor sounds a beep and returns to the calling program. If not equal, the Monitor prints "ERR" through COUT before sounding the beep and returning.

If you want to have the calling program determine whether the tape was read successfully or not, then some special actions must be taken. One method is to compare the contents of CH ($24) before the tape read with the contents after. If they are equal, ERR was not printed to the screen. If the cursor horizontal position (CH) has changed across the call to READ, then ERR must have been written to the screen. If this condition is encountered, the program can then ask the operator to position the tape and signal the program for another attempt at reading the record. Caution: If CSWL,H points to a printer card or other routine which does not output to the screen, CH will not be incremented by the output of "ERR".

CASSETTE INPUT/OUTPUT INTERNAL ROUTINES

The following entry points/routines functions are described , but not documented in sufficient detail for call by user program. For some of them, timing is critical and the documentation for using them would depend on how they were to be used.

HEADR

$FCC9 64713 -823

This routine writes the synchronization monotone which is the first part of every tape record. When the WRITE routine calls HEADR, it loads a $4Ø into the A-reg causing a 1Ø second header to be written. The READ routine also calls HEADR to delay from first detection of data coming in from the tape to the first point at which reading for Ø/1 detection begins. READ loads the A-reg with a $16 before calling HEADR so the delay for hardware settling is set to about 3.5 seconds. This routine is not called by BASIC or APPLESOFT, but it is used by the Programmer's Aid #1 Tape Verify routines which read the tape and compare the data to memory instead of storing the data into memory.

RD2BIT

$FCFA 64762 -774

This routine causes looping with decrementing of the Y-reg until the hardware has indicated two transitions of the tape input register. The routine RDBIT is called twice for this purpose. Contents of the Y- reg on return compared with contents on entry indicate the length of time it took for the transitions.

This routine is called from within the Monitor by the READ routine, to delay entering data transfer mode until tape input is available. READ calls HEADR for the 3.5 second delay on return from its call to RD2BIT. This routine is also called from APPLESOFT and from the Tape Verify and Shape Table Load programs in the Programmer's Aid #1.

RDBIT

$FCFD 64765 -771

This routine loops with decrementing of the Y-reg while testing the tape input register for transition from zero to one or one to zero. Bit value of zero or one is then determined from the residual count in the Y-reg. This routine is called from within the Monitor routines RD2BIT and READ. It is also called by Programmer's Aid #1 Tape Verify.

RDBYTE

$FCEC 64748 -788

This routine calls RD2BIT as required in order to assemble a byte of information from the tape. It then returns to caller with the byte in the A-reg. In addition to being called from the Monitor READ routine, it is also called by Shape Table Load in Programmer's Aid #1.

WRBIT

$FCD6 64726 -81Ø

This routine accomplishes writing a bit to the tape when called by either the HEADR routine or the WRBYTE routine.

WRBYTE

$FEED 65261 -275

When called to write a byte to the tape, this routine uses WRBIT to write ten bits to the tape. The only caller is WRITE in the Monitor.

PADDLES, BUTTONS & ANNUNCIATOR I/O

The Apple II has a Game I/O connector with hardware support for four digital outputs, three digital inputs, and four analog inputs (called paddles). The Monitor reads the paddles by writing a strobe to start the paddle timer and then reading the selected paddle timer and incrementing the Y-reg until that timer comes true. The result of the read is in the Y-reg. Monitor support for digital outputs or digit inputs is not required. Access to the digital I/O ports is gained by PEEKing or POKEing the appropriate address, or by LDx or STx if machine language is used. The Autostart Monitor does initialize the digital output ports (annunciators) on any RESET key interrupt. ANØ and AN1 are initialized to the clear (TTL LO) condition by reference to addresses $CØ58 and $CØ5A. AN2 and AN3 are initialized to the set (TTL HI) condition by reference to addresses $CØ5D and $CØ5F.

To use the Monitor support to read the setting of a paddle, JSR to

PREAD FB1E 64286 -125Ø

with paddle number (Ø-3) in X-reg, and on return the "value" of the paddle will be found in the Y-reg. The A-reg is destroyed in the process. (APPLESOFT and BASIC support paddle reading, so setting of X and looking at Y is not required there.)

Direct reading of the paddles may be accomplished by accessing the paddle trigger to start all paddle timers and then reading the appropriate paddle input address repeatedly while counting until the value read from the paddle address no longer has the $8Ø bit set.

CAUTION: After reading a paddle, let some time go by before reading another paddle or incorrect results may be a problem. When the paddle trigger is strobed, all the timers start. If the first paddle you read has a low value, on going back quickly to read another paddle the transition you see may be from the first paddle trigger instead of the second. See the sample program in the section "Use of Control-Y with Parameters". Another solution is to do a read of a fake paddle between real readings.

GAME I/O HARDWARE ADDRESS TABLE

Game I/O Hardware Address	Hex Addr	+Dec Addr	-Dec Addr	Action/Comments
Start Paddle Timers.	CØ7Ø	49264	-16272	
Paddle Ø timer.	CØ64	49252	-16284	Negative until
Paddle 1 timer.	CØ65	49253	-16283	timer
Paddle 2 timer.	CØ66	49254	-16282	expires.
Paddle 3 timer.	CØ67	49255	-16281	
Paddle Ø switch.	CØ61	49249	-16287	Negative
Paddle 1 switch.	CØ62	4925Ø	-16286	indicates
Paddle 2 switch.	CØ63	49251	-16285	button pushed.
Clear Annunciator Ø output.	CØ58	4924Ø	-16296	POKE/STore
Set Annunciator Ø output.	CØ59	49241	-16295	zero
Clear Annunciator 1 output.	CØ5A	49242	-16294	to
Set Annunciator 1 output.	CØ5B	49243	-16293	appropriate
Clear Annunciator 2 output.	CØ5C	49244	-16292	address.
Set Annunciator 2 output.	CØ5D	49245	-16291	
Clear Annunciator 3 output.	CØ5E	49246	-1629Ø	
Set Annunciator 3 output.	CØ5F	49247	-16289	

WAIT ROUTINE

The WAIT routine consists of a loop within a loop, constructed in such a manner that the length of time spent in the loop varies geometrically with the entry A-reg. A call to this routine will cause a loop for a predictable length of time, such as is used by the Monitor with regards to using the speaker as a bell. It may be usable, for example, in writing data to a lower speed device like a printer or a typewriter.

WAIT $FCA8 6468Ø -856

Anaylsis of the code indicates that the time between the call WAIT (JSR) and the end of the RTS of WAIT is approximately
2.5A**2 + 13.5A + 13 machine cycles of 1.Ø23 microseconds.
where A equals the contents of the accumulator.

An alternative formula is
TIME IN MICROSECONDS = (2.5 * (A^2) + 13.5 * A + MC) * MS
where A = contents of accumulator
MC = 13 machine cycles
MS = 1.Ø23 microseconds

The following table indicates delay times in the WAIT routine for a number of values of the A-reg on entry.

WAIT ROUTINE DELAY TIMES

A-reg (Dec.)	Time in seconds	A-reg (Dec.)	Time in seconds	A-reg (Dec.)	Time in seconds
1	.ØØØØ29667	49	.ØØ683Ø571	137	.Ø499Ø7Ø55
2	.ØØØØ5115	5Ø	.ØØ7Ø97574	138	.Ø5Ø624178
3	.ØØØØ77748				
4	.ØØØ1Ø9461	53	.ØØ7929273	15Ø	.Ø59628624
5	.ØØØ146289	54	.ØØ8216736	151	.Ø6Ø412242
6	.ØØØ188232	55	.ØØ85Ø9314		
7	.ØØØ23529	56	.ØØ88Ø7ØØ7	162	.Ø6936963
8	.ØØØ287463	57	.ØØ91Ø9815	163	.Ø7Ø214628
9	.ØØØ344751	58	.ØØ9417738		
		59	.ØØ973Ø776	174	.Ø79847196
17	.ØØØ987195	6Ø	.Ø1ØØ48929	175	.Ø8Ø753574
18	.ØØ1Ø9Ø518				
19	.ØØ1198956	73	.Ø1465Ø383	184	.Ø89141151
		74	.Ø15Ø4Ø146	185	.Ø9ØØ98679
25	.ØØ1956999	75	.Ø15435Ø24		
26	.ØØ21Ø1242			195	.Ø99955284
		85	.Ø19665129	196	.1ØØ969Ø77
31	.ØØ2899182	86	.Ø2Ø116272		
32	.ØØ3Ø74115			2Ø4	.1Ø9263561
		96	.Ø249Ø9Ø27	2Ø5	.11Ø323389
36	.ØØ3824997	97	.Ø25416435		
37	.ØØ4Ø255Ø5			218	.124566618
		1Ø5	.Ø29659839	219	.125698Ø56
41	.ØØ4878687	1Ø6	.Ø3Ø213282		
42	.ØØ51Ø477			239	.1494ØØ966
		122	.Ø39764Ø1	24Ø	.15Ø639819
45	.ØØ58137Ø9	123	.Ø4Ø4Ø44Ø8		
46	.ØØ6Ø6Ø252			255	.169836414

USE OF CONTROL-Y WITH PARAMETERS

In the APPLESOFT manual there is a caution that if one paddle is read another should not be read too quickly. Following is a machine language program with which the interference between the paddles can be demonstrated.

Initiate this program by entering the Monitor command xxxxY, where xxxx is a number representing the amount of delay to use between reading paddle Ø and reading paddle 1, and Y represents control-Y. The Monitor command "control-Y" causes a JMP to location $Ø3F8 at which location we place a JMP to the beginning of the program.

As the Monitor scans the input command line, the value of the hex digirs is placed in page zero locations A1L,H ($3C-3D) for our use.

PADDLE INTERFERENCE—SAMPLE PROGRAM

```
Ø3F8  JMP  $2ØØØ

2ØØØ  LDA  $#CØ     Set counter for 64 samples to run
2ØØ2  STA  $4        before clearing screen and starting over.
2ØØ4  LDA  $3C      Pick up low part of entered count from A1L
2ØØ6  STA  $1Ø       and store it for repeated use.
2ØØ8  LDA  $3D      Pick up high part of entered count from A1H
2ØØA  STA  $11       and store it for repeated use.

2ØØC  LDA  $1Ø      Pick up low part of count:
2ØØE  STA  $12       store it in counter for this pass,
2Ø1Ø  LDA  $11       and also high part.
2Ø12  STA  $13
2Ø14  LDX  $#Ø      Set X for paddle Ø read.
2Ø16  JSR  $FB1E    Call paddle read.
2Ø19  STY  $Ø       Store paddle Ø result in location Ø.

2Ø1B  DEC  $12      Count down delay loop low byte:
2Ø1D  BNE  $2Ø1B     when zero, count down high byte.
2Ø1F  DEC  $13
2Ø21  BMI  $2Ø1B    Stay in the loop until high goes minus.

2Ø23  LDX  $#1      Set X for paddle 1 read.
2Ø25  JSR  $FB1E    Call paddle read.
2Ø28  STY  $1       Store paddle 1 result in location 1.

2Ø2A  LDA  $Ø       Pick up paddle Ø value.
2Ø2C  JSR  $FDDA    Print it as a hex value.
2Ø2F  LDA  $#AØ     Pick up a blank to print.
2Ø31  JSR  $FDED    Print the blank.
2Ø34  LDA  $1       Pick up paddle 1 value.
2Ø36  JSR  $FDDA    Print it as a hex value.
2Ø39  JSR  $F948    Print three blanks.

2Ø3C  INC  $5       Delay for awhile to keep paddle 1 read
2Ø3E  BNE  $2Ø3C     from upsetting paddle Ø results.

2Ø4Ø  INC  $4       Is it time to clear screen and restart?
2Ø42  BNE  $2ØØC     NE means no, go back and sample again.

2Ø44  LDA  $#Ø      Wait a while before clearing screen.
2Ø46  STA  $4
2Ø48  STA  $5

2Ø4A  INC  $4
2Ø4C  BNE  $2Ø4A
2Ø4E  INC  $5
2Ø5Ø  BNE  $2Ø4A
2Ø52  JSR  $FC58    Clear the screen.
2Ø55  LDA  $#CØ     Restore the per screen counter,
2Ø57  STA  $4
2Ø59  BNE  $2ØØC     and go one more big round.
```

REGISTERS FOR BASIC MONITOR CALLS

Many of the entry points specified in this book require presetting of registers for proper operation. Following is a sample program, written for APPLESOFT, which uses Monitor calls for conversion from decimal to hex.

The theory behind the operation is that on a Monitor G command, the registers are loaded from the SAVE area before going to the location specified in PCL,H. Thus, by poking destination address into PCL,H and the required register contents into XREG, YREG, an entry point in the Monitor Go command processor can be used to pass the registers to a selected routine.

DECIMAL TO HEX CONVERSION

APPLESOFT SAMPLE PROGRAM

```
10   REM CONVERT DECIMAL INPUT TO HEX OUTPUT
100  INPUT "ENTER NUMBER ";A     Read the input.
110  IF A=99999 THEN END         Provide a way to end the program.
150  C% = A / 256                Isolate the high byte.
200  POKE 71,C%                  Set YREG for PRNTYX call.
300  B% = A / 256                Get remainder from A/256.
310  B = B% * 256                For low byte (XREG) POKE.
320  B% = A - B
350  POKE 70,B%
400  POKE 59,249                 Set PCH to $F9.
500  POKE 58,64                  Set PCL to $40.
550  PRINT                       Print a blank line.
600  CALL 65209                  Entry point in GO processor is FEB9.
650  PRINT                       Print a blank line.
700  GOTO 100                    Go around for another number.
```

STEP AND TRACE PECULIARITIES

The Step and Trace functions in the Old Monitor incorrectly display register contents under some circumstances. The STEP routine detects and gives special attention to JSR, RTS, JMP, JMP indirect, RTI, and BRK instructions. In each case, the register contents are displayed from the SAVE area at $45-49. However, there is no SAVE call after "execution" of these instructions, as there is for normally traced instructions, so the registers displayed are those present in the SAVE area before execution of this instruction.

Therefore, on JSR and RTS, the displayed contents of the S-reg are incorrect. On the first instruction after a JSR or RTS, the S-reg displays correctly, unless that also is an RTS or JSR.

The Step and Trace routines are not incorrect in handling of a BRK instruction. That is, the address displayed for the BRK is correct, instead of being off by two bytes, because the BRK is detected by the STEP routine instead of being executed by the 6502.

Although step and trace can be very helpful for some program debugging tasks, they cannot be used in tracing calls to the Monitor (generally including "print" output) or for programs which use A1L,H thru A4L,H.

Because of the lack of "CLD" at PCADJ ($F953), incorrect addresses will be displayed if you set decimal mode (SED) within the program being traced or stepped.

apple computer inc.

10260 Bandley Drive
Cupertino, California 95014
(408) 996-1010

950-0018-A